AGARD-LS-143

AGARD-LS-143

AGARD

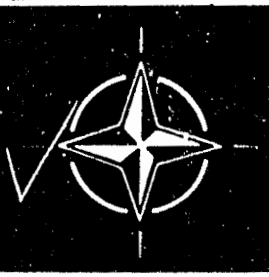
ADVISORY GROUP FOR AEROSPACE RESEARCH & DEVELOPMENT

7 RUE ANCELLE 92200 NEUILLY SUR SEINE FRANCE

AGARD LECTURE SERIES NO.143

Fault Tolerant Hardware/Software Architecture for Flight Critical Function

NORTH ATLANTIC TREATY ORGANIZATION



DISTRIBUTION AND AVAILABILITY ON BACK COVER

AGARD-LS-143

NORTH ATLANTIC TREATY ORGANIZATION

ADVISORY GROUP FOR AEROSPACE RESEARCH AND DEVELOPMENT

(ORGANISATION DU TRAITE DE L'ATLANTIQUE NORD)

AGARD LECTURE SERIES No.143

FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION

The material in this publication was assembled to support a Lecture Series under the sponsorship of the Guidance and Control Panel and the Consultant and Exchange Programme of AGARD presented on 1—2 October 1985 in Edwards, USA, 17—18 October 1985 in Copenhagen, Denmark, and 21—22 October 1985 in Athens, Greece.

THE MISSION OF AGARD

The mission of AGARD is to bring together the leading personalities of the NATO nations in the fields of science and technology relating to aerospace for the following purposes:

- Exchanging of scientific and technical information;
- Continuously stimulating advances in the aerospace sciences relevant to strengthening the common defence posture;
- Improving the co-operation among member nations in aerospace research and development;
- Providing scientific and technical advice and assistance to the North Atlantic Military Committee in the field of aerospace research and development;
- Rendering scientific and technical assistance, as requested, to other NATO bodies and to member nations in connection with research and development problems in the aerospace field;
- Providing assistance to member nations for the purpose of increasing their scientific and technical potential;
- Recommending effective ways for the member nations to use their research and development capabilities for the common benefit of the NATO community.

The highest authority within AGARD is the National Delegates Board consisting of officially appointed senior representatives from each member nation. The mission of AGARD is carried out through the Panels which are composed of experts appointed by the National Delegates, the Consultant and Exchange Programme and the Aerospace Applications Studies Programme. The results of AGARD work are reported to the member nations and the NATO Authorities through the AGARD series of publications of which this is one.

Participation in AGARD activities is by invitation only and is normally limited to citizens of the NATO nations.

Published September 1985

ISBN 92-835-1510-2

Printed by Specialised Printing Services Limited
40 Chigwell Lane, Loughton, Essex IG10 3TZ

LIST OF SPEAKERS

Lecture Series Director: Mr G.L.Hartmann
Honeywell Inc. MN 17-2332
Systems and Research Center
Aerospace and Defense Group
2600 Ridgway Parkway
P.O. Box 312
Minneapolis, Minnesota 55440
USA

SPEAKERS

Dr T.Anderson
Computing Laboratory
University of Newcastle Upon Tyne
Claremont Tower, Claremont Road
Newcastle Upon Tyne NE1 7RU
UK

Mr H.Belmont
Engineering Specialist
Northrop Aircraft Co.
1, Northrop Avenue
Hawthorne, CA 90250
USA

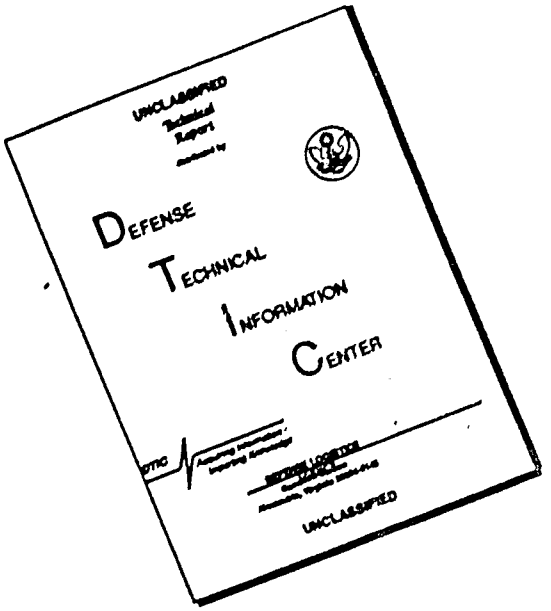
Dr W.Heimerdinger
Honeywell SRC — Computer Sciences
260 Ridgeway Parkway (MN17—2351)
Minneapolis, MN 55440
USA

Mr G.M.Papadopoulos
Massachusetts Institute of Technology
Dept. of Electrical Engineering and
Computer Science
545 Technology Square
Cambridge, MA 02139
USA

Mr D.R.Powell
Chargé de Recherche au CNRS
d'Analyse des Systèmes
CNRS
7, Av. du Colonel Roche
31077 Toulouse CEDEX
France

Mr A.D.Hills
Engineering Manager
Flight Controls Division
GEC Avionics Limited
Airport Works
Rochester, Kent ME1 2XX
UK

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

CONTENTS

INTRODUCTION TO
LECTURE SERIES NO. 143
FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURES FOR FLIGHT CRITICAL FUNCTIONS

Gary L. Hartmann

Honeywell Inc.
Systems and Research Center
2600 Ridgway Parkway P.O. Box 312
Minneapolis, Minnesota USA 55440

Modern weapon systems, driven by escalating performance demands, are becoming complex and sophisticated. Demands for higher accuracy, improved reliability/survivability, all-weather operation, and more automation are placing increased emphasis on the control function. Nowhere is this increased emphasis more evident than in the control functions required in advanced aircraft systems. Due to the expanded role of automation many functions are becoming flight-critical (i.e., loss of this function is catastrophic).

Flight critical architectures are more complex than fault-tolerant computers. In addition to airborne computers, overall reliability depends on proper design and management of:

- Sensors and their interfaces
- Actuation elements
- Data communication among the distributed elements

Previous NATO-AGARD publications have dealt with related aspects of this subject. Lectures Series No. 109 [reference 1] discussed redundancy management aspects of flight control with some detail on sensor management and analytical redundancy techniques. AGARD publications such as reference 2 cover areas related to integrity in electronic flight control systems. This lecture series covers experience with flight tested fly-by-wire systems as well as issues in redundancy management of synchronous and asynchronous systems. It specifically addresses software fault tolerance, actuation fault tolerance, reliable data communications, and multi-computer operation using the Ada language.

First Day

The first paper presents a description of two recent GEC Avionics systems:

- A310 Slats and Flaps Control System
- Jaguar FBW Demonstrator Flight Control System

Particular reference is made to the architectures of the computers and embedded software. The system design requirements, especially those relating to integrity and availability, are presented. Emphasis will be placed on the reasons for selecting dissimilarity, as an implementation philosophy, for the A310 system, as against the multiple similar elements in the Jaguar architecture.

The paper provides a brief description of the design and development programs for the two computer units with emphasis on lessons learned, especially in the software areas. The aspects of the system involved with maintainability and reliability are detailed and current in-service experience discussed. Conclusions highlight lessons learned from these successful fly-by-wire programs.

The second paper addresses a fundamental issue in managing redundany elements -- should the redundant channels be synchronized or allowed to run asynchronously? While asynchronous systems may initially appear attractive due to the "uncoupling" of the channels, the cross-channel interactions are much greater than what might be expected. Both synchronous and asynchronous systems share the burden of cross-channel consistency maintenance, the requirement that inputs and internal states are not allowed to diverge. In fact, consistency maintenance often dominates the design process in a correctly designed system. In asynchronous systems consistency maintenance takes the form of cross-channel equalization along with techniques for handling discrete changes in operating mode.

Consistency maintenance and resoultion techniques are presented for both types of systems. The class of synchronous systems are further decomposed into clock synchronous and data synchronous. The relative strengths and merits of all three approaches are contrasted. It is concluded that, based on the increasing complexity of future systems, synchronous systems will be preferable.

The third paper will present a case for the adoption of design-fault tolerant techniques in practical software systems. Fault tolerance has an established role in detecting and masking component faults in hardware systems, and is being advocated as a defense against design deficiencies which can plague software.

A critical application area for computer systems is that of real-time control.(e.g., in avionics). Strong support for the utility and effectiveness of software fault tolerance in these systems is provided by the results of an experimental project at the University of Newcastle upon Tyne. Techniques were demonstrated with a realistic implementation of a control system. Reliability data was collected by running this system in a simulated tactical environment for a variety of action scenarios. Analysis of the data showed that reliability was significantly enhanced by the use of software fault tolerance.

The fourth paper addresses information transfer in flight critical systems. First-generation avionic data transmission systems (MIL-STD-1553, ARINC 429, GAM-T-101 (Digibus)) were contemporary to what have now become known as local area networks. They were mainly motivated by the desire to decrease the amount of wiring used for equipment interconnection. All of these systems are relatively slow by modern standards and feature little or no fault-tolerance features (all use centralized control).Since 1976, the Dependable Computing System Design and Validation group at LAAS has been engaged in two research projects concerned with architectures and protocols for the next generation of avionic data transmission systems. The increased reliance on digital techniques in present-day aircraft, will require order-of-magnitude higher data rates and improved dependability.

This paper presents several techniques for realizing multiple path broadcast media along with fully-decentralized control protocols. Specially-developed Carrier Sense Multiple Access (CSMA) techniques featuring low error latency and station autonomy while ensuring the bounded access times required of real-time systems are described.

The first day of the lecture series concludes with part I of a paper on the use of the Ada programming language in flight critical applications. Ada has been mandated as the programming language to be used for future U.S. Department of Defense embedded computer applications. It was designed primarily for embedded computer system applications, and incorporates a number of features to enhance program clarity and to improve error containment.

Second Day

The second day of this lecture series begins with a paper addressing design issues in data synchronous systems.

In data synchronous systems, the outputs of all correctly operating redundancy channels are guaranteed to bit-for-bit agree, independent of whether the channels are clock or frame synchronous. Data synchronous systems offer a general form of fault tolerant processing capable of correctly supporting a very general class of programs.

Using simplified dataflow models, the various aspects of the design of correct data synchronous systems are examined in detail. These include: Source consistency, the requirement that all correctly operating channels receive precisely the same inputs. Task synchronization, the problem of keeping the time skew between channels within predetermined bounds, as well system initialization, sparing, and transient recovery. The unsolved problem of latent faults is also presented along with the need for self-test heuristics. Sequential fault tolerant and parallel fault tolerant approaches are contrasted for systems requiring protection from multiple faults. Both hardware and software solutions to these problems are given, emphasizing system performance and economy.

The next paper address the important (and sometimes neglected) interface with aircraft actuation systems. Digital technology has effectively been applied to the computational aspects of flight control systems but must interface with traditionally analog power elements (e.g., hydraulic servo actuators). To evaluate this interface, the flight control actuation subsystem was defined in terms of three equipment elements (control processor, servo processor and servo actuator complex) and their communication interfaces.

A set of design issues were identified with respect to these elements. These issues included functional task assignments and physical locations. In addition, requirements for high temperature electronics, feasibility of digital mechanization, and fault detection/redundancy management were design considerations. Of the candidate flight control system configurations studied, one which provided the highest potential for fault tolerance was a triplex, active-on-line system with self-checking computer pairs in each channel for control law and servo actuator processing. The latter set was located at each actuator and was connected to all contol law processors by means of a digital serial data bus, thus creating a voting plane at the actuator.

The next paper presents a methodology for validating the functions and reliability of a fly-by-wire system. Recommendations for using finite state machines to make the System Specification precise and complete are made. Examples are drawn from several flight control designs to illustrate the use of finite state machines.

In validation of system reliabilty, fault tree analyses is combined with a finite state representation of the redundancy management to establish system level reliability tests. This method is illustrated using design trades from a distributed multi-computer architecture.

The final paper is part II of the use of Ada in multiple computer systems. Two ongoing projects at Honeywell's Systems and Research Center illustrate the range of options in distributing Ada software on multiple computers.

REFERENCES

(1) AGARD Lecture Series 109, Athens, Rome, London, October 13-21 1980

(2) Integrity in Electronic Flight Control Systms, AGARDograph No. 224, Published April 1977.

DIGITAL FLY-BY-WIRE EXPERIENCE

Mr. A.D. Hills
Engineering Manager
Flight Controls Division
GEC Avionics Ltd
Airport Works, Rochester,
Kent, England.

SUMMARY

A description of two recent GEC Avionics (GAv) systems is included

1) A310 Slats and Flaps Control System

2) Jaguar FBW Demonstrator Flight Control System

Particular reference is made to the architecture of the computers and embedded software.

Data is included on the two different system design requirements, especially those relating to integrity and availability. Emphasis is placed on the reasons for selecting dissimilarity, as an implementation philosophy, for the A310 system, as against the multiple similar Jaguar demonstrator architecture.

The paper then provides a brief description of the design and development programmes for the two computer units with emphasis on lessons learned, especially in the software areas.

The aspects of the system and computer design involved with maintainability and reliability are detailed and current in-service experience discussed where applicable.

Conclusions are drawn particularly on dissimilarity, highlighting lessons learned from these successful FBW programmes.

A potential method of providing software fault tolerance within a dissimilar implementation is discussed and preliminary results of a GEC research programme in this area provided.

INTRODUCTION

The two recent GEC Avionics Fly-by-Wire (FBW) programmes addressed in this paper were conceived for two very different system and integrity requirements.

One, the Jaguar FBW system, was designed and developed as a production standard demonstrator control system to investigate and prove advanced control concepts for future fighter aircraft. The A310 Slat and Flap system however, is a FBW implementation of a secondary flight control system for a commercial twin aisle passenger transport now in scheduled service.

The essential differences both in certification and system requirements led the two GEC Avionics design teams to adopt two different systems solutions and hence computer architectures.

A310 SLATS AND FLAPS SYSTEM REQUIREMENTS AND DESCRIPTION

The Slat and Flap Control system for the Airbus Industrie A310 aircraft is a digitally implemented electrical control system which has no mechanical linkage between the cockpit mounted Slat/Flap control lever and the respective hydraulic actuator. The system was designed in collaboration with Liebherr - Aero-Technik GmbH for MBB GmbH, Bremen, West Germany, who form part of the Airbus Industrie consortium. This FBW implementation was selected for the following reasons:-

* Weight saving.
* Repeatabiliity of very accurate surface deployment without rigorous mechanical maintenance.
* Maintainability and fault diagnosis improvements over a mechanical system.
* Incorporation of system protection features within electronics leading to the use of lighter screw jacks.
* Flexibility for modification and inclusion of pilot work-load reduction features such as 'Slat baulk'.

The integrity requirements for this system can be summarised as:-

* Failure to operate surface when commanded but failure indicated to crew.	$< 10^{-5}$ per flight hour.
* Failure to operate surface when commanded and failure not indicated to crew. * Uncommanded surface movement. * Asymmetric deployment of surfaces.	$< 10^{-9}$ per flight hour.

The factors which were considered in the selection of the computer and software architecture required to support this integrity and availability requirement were as follows:-

* Extent of task - could it be contained within a design using the then available avionic grade microprocessors?
* Certification authority opinion on the use of redundant monitoring or limiting devices.
* Concern over analysis of software integrity and LSI device failure modes.

These last two factors led to the use of dissimilarity both in hardware and software as part of the final architecture.

System Description

Flight crew surface deployment commands are transmitted as redundant electrical discrete signals from the cockpit Slat/Flap control lever to the computers.

Figure 1 shows the interface of one channel of the flaps control system, the slats system being similar in architecture.

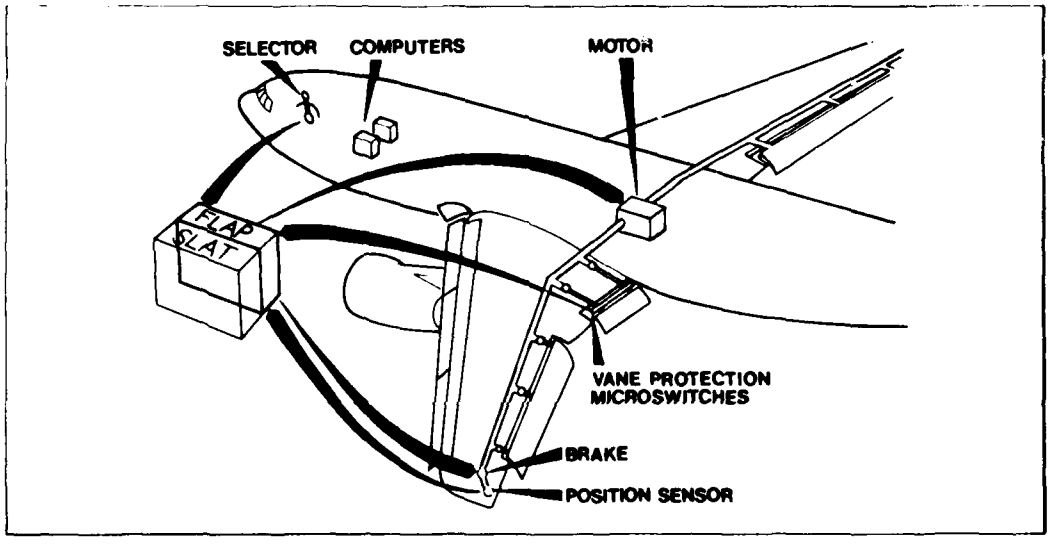


Figure 1 Interface of Electronic Control for Flap Drive

These commands are verified and then used to compute the direction and speed of surface movement, if required. By using incidence information from both air data sources, these pilot commands may be modified to prevent inadvertent full slat retraction above an incidence threshold. A cockpit warning is provided if this protection feature becomes active.

The facility also exists to limit flap deployment as a function of airspeed. This function is implemented in the derivative GEC Avionics' Slats and Flaps Control System for the Airbus Industrie A300 - 600 aircraft.

Position feedback is provided by synchros attached to the output shafts of the slat and flap hydraulic motors. Additional synchros mounted at each end of the transverse torque shafts provide data for system asymmetry and speed monitoring.

Upon detection of a system asymmetry, uncommanded surface movement or other critical failures, electro-hydraulic brakes are applied at the end of torque shafts in order to lock the system. This brake application, which must occur within approximately 40ms of critical failures occurring, requires the agreement of both computing channels associated with the appropriate slat or flap function.

System and Computer Architecture

The deployment of surfaces and the intensive system monitoring are controlled by two identical digital computers. Each of these two computers contains two further identically implemented but independent channels, one channel dedicated to each of the slat or flap functions. Each channel drives the solenoid valves of its associated hydraulic motor, which, via a differential gear arrangement, tranverse torque shafts and rotary actuators, operates the respective slat or flap surface. Failure of a channel causes a pressure off brake to be applied to the relevant side of the differential and surface deployment is then available at half the normal speed.

To complete the system monitoring concept, a duplex computing arrangement was selected for each of the independent slat or flap channels.

The outputs from the two duplex lanes are consolidated by use of combinational logic techniques. This ensures that no drive commands can be generated by a channel unless both independent dissimilar lanes agree. For certain critical outputs, eg, electro-hydraulic wing tip brake operation, either lane has the capability of setting its relevant channel output. Monitoring is carried out over a number of computing frames to avoid nuisance warnings.

Figure 2 illustrates the computer architecture and shows the consolidation arrangement between the dual dissimilar lanes.

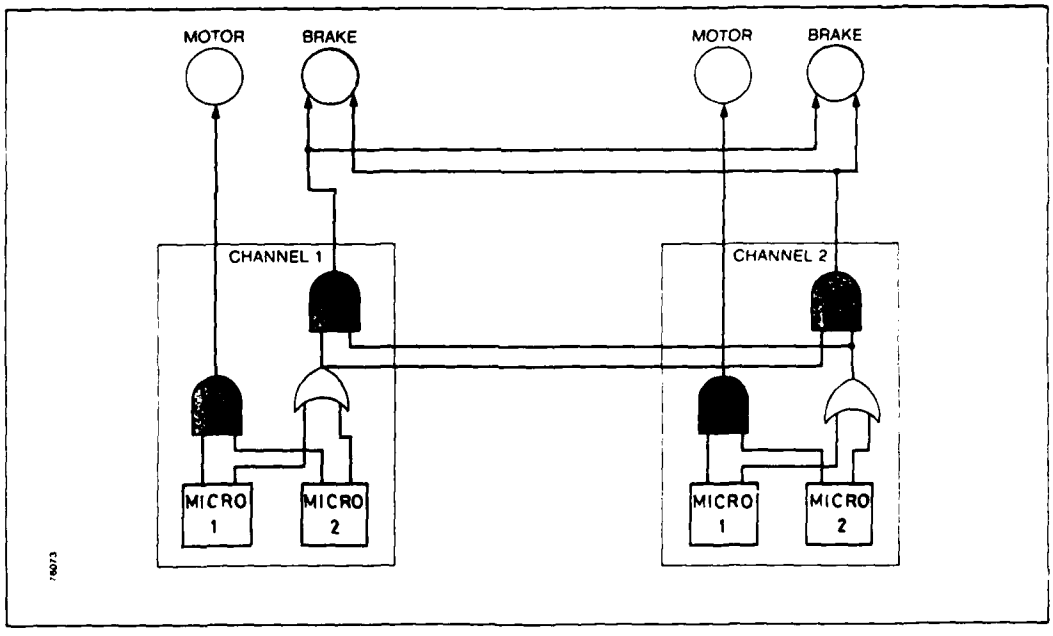


Figure 2 Computer Architecture Schematic

In order to protect the system against common design errors generating uncommanded surface deployment, the concept of dissimilarity was employed. The two computing lanes within a channel were therefore designed as follows:

* Each lane of a channel is implemented using a different microprocessor procured from different manufacturers.

  This decision reduces the FMEA requirement on a chosen common device and assists in the achievement of dissimilar software by enforcing dissimilar assembler statements.

* Each lane contains its own clock and timing signals and operates asynchronously to the other lane, both in hardware and software terms.

* Store maps implemented for each lane are configured differently.

* Care was taken in the allocation of commands within output words to ensure that similar discrete commands did not occupy identical bit positions within words. This is necessary since the monitoring system of cross comparing critical output commands requires output discretes from one lane to be mapped as inputs to the other lane.

Software Architecture

The software philosophy adopted for the Aileron Slat and Flap Computer is one of dissimilarity from the specification through the source statements and host systems to (E)PROM file formats. This decision to utilise a continuous cross comparison between dissimilar implementations as protection against a dormant error was taken to circumvent the problem of demonstrating very high integrity levels in similar software-based systems.

A separate software requirements document (SRD) was produced for each lane. Having produced the SRD, the development procedure follows the normal path, in each lane, of top-down analysis to produce a modular structure and then to design and generate code for each module using assembler statements.

To avoid the remote but credible possibility of a common host computer error introducing faults into the software two different host systems were used to assemble code, one for each lane.

At this stage, instead of embarking on extensive module testing, the approach that has been taken is to assemble the software for each lane and then to perform hardware/combined software integration testing. Experience with this technique has slightly modified this approach and integration testing is now first conducted on individual lanes before integrating both dissimilar lanes together. This partial integration reduces the number of potential error sources and hence speeds the overall task.

The key features of the dissimilar software development process are shown in Figure 3.

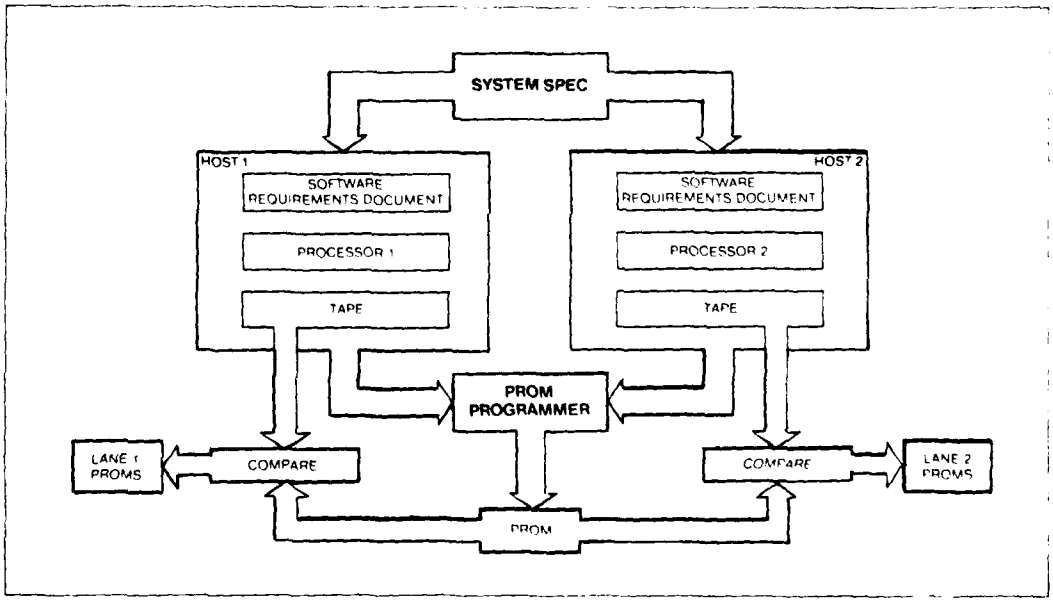


Figure 3 Dissimilar Software - Key Features

JAGUAR FLY-BY-WIRE SYSTEM REQUIREMENTS AND DESCRIPTION

The prime requirement of the demonstrator FBW Jaguar Integrated Flight Control System (IFCS) was to provide full authority control of the aircraft's five primary control surfaces namely

Left and right tailplane
Rudder
Left and right spoilers

as shown in Figure 4; - the primary control surfaces being shown cross hatched.

The multiple redundant design philosophy of the FBW Jaguar IFCS was largely dictated by the overall requirements:-

* Overall system loss probability (up to and including the first stage of the control surface actuation) should be no greater than 10^{-7} per hour.

* The system should be able to survive any two electrical failures.

* The system should use electro-hydraulic first stage actuation with the constraints that only two independent hydraulic supplies would be available, and hydraulic failures could not be alleviated by any form of interconnection between the two hydraulic supplies.

* The system should be able to survive a hydraulic system failure followed by an electrical failure, or an electrical failure followed by a hydraulic failure.

* The system should in general rely on majority voting of the redundant elements for failure survival rather than self monitoring within each of the redundant elements.

* Similar redundant digital implementation (both hardware and software) should be adopted without any reliance on any back up flight controls (e.g. mechanical or simple analogue links).

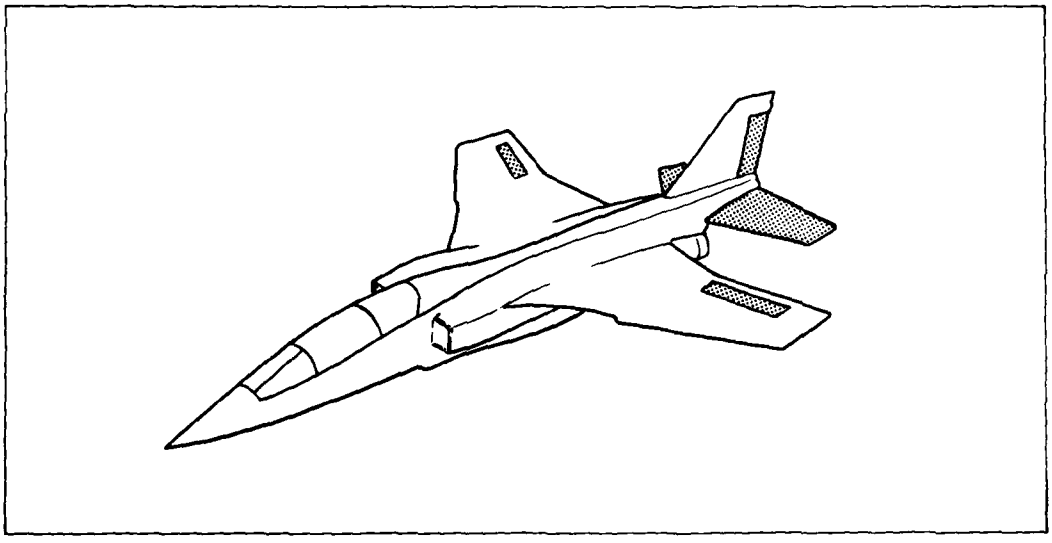


Figure 4 FBW Jaguar showing Primary Control Surfaces

System Description

The first of these requirements was based on the desire to develop a full time FBW system which could be shown to be at least as reliable (in terms of overall system loss probability) as the mechanical control linkages which it replaced. For this reason the particular double hydraulic failure case did not need to be considered as this was a common denominator (in terms of non-reversible tandem power actuation) between the FBW system and its traditional mechanically linked equivalent.

That the second requirement is a by-product of the first is evident when one considers that the failure probability of a single lane of a flight control system of the required complexity in a military environment is likely to be of the order of 10^{-3} per hour. In other words a single fail operational triplex system, which with this lane failure rate would have a second failure probability of 3.10^{-6} per hour would clearly not meet the above system loss probability requirement. It was therefore decided to opt for quadruplex computers to perform the flight control computing tasks with quadruplex primary input sensors (i.e. pilot commands, aircraft angular rates, and control surface positions). Such an arrangement is able to survive two sequential input sensor failures by means of appropriate majority voting and failure rejection logic within the computers.

In view of the requirements above the first stage actuation clearly needed to be either quadruplex (with two sub-actuators per hydraulic supply) or sextuplex (with three sub-actuators per hydraulic supply). The quadruplex configuration had the attraction of a tidy (one-to-one) interface with the quadruplex flight control computers (FCCs).

However in order to meet requirement for surviving electrical and hydraulic faults, some form of actuation monitoring and sub-actuator lane isolation mechanism would have been necessary, and would have placed a heavy reliance on the ability of the system to reject an electrical failure (following a hydraulic system failure) in an acceptably short time to prevent uncontrolled surface movement and possible catastrophic aircraft divergence.

It was therefore decided to 'play safe' and incorporate sextuplex (or duo triplex) first stage actuation. The feature of such a first stage actuation system is that with such a high level of redundancy, lane failures can be survived by the process of failure absorption i.e. the actuation elements do not need any hardware associated with either monitoring or isolation. Any two sub-actuators or sub-actuator input failures can be survived by virtue of the fact that there are always more 'good' sub-actuators than 'bad' ones. A hydraulic failure is absorbed because the three sub-actuators operated by that hydraulic system are not able to oppose the three (or two after a further failure) correctly operating sub-actuator lanes.

The remaining problem was how to interface four FCCs with each of the six lane first stage actuators in such a way that one or two FCC failures could not propagate to more than two lanes of the six lane actuators. The solution which was adopted was to interface the four FCCs on a one-to-one basis with four out of the six lanes of each first stage actuator, and to drive the remaining two lanes of each first stage actuator with independently voted versions of the FCC output drive signals. These two additional voting nodes required two further computers, each with appropriate segregation of data transmission and power supplies etc., in order to eliminate any possibility of inter-lane fault propagation between the six parallel redundant output interfaces. The resultant Jaguar FBW system configuration is shown in Figure 5.

In addition to the quadruplex set of primary input sensors, sensors of lower redundancy are used for those functions which may be necessary for optimum performance but which are not necessary for safe flight. These are as follows:-

Dynamic pressure	triplex
Static pressure	triplex
Lateral acceleration	duplex
Angle of attack	duplex
Sideslip	duplex
Autopilot sensors	simplex

The dynamic and static pressure inputs are used for gain schedules etc. In the event of a second similar failure the system reverts to safe fixed values. The fail safe lateral acceleration input is provided as an adjunct to the prime yaw rate and sideslip angle rudder control terms. The location of the angle of attack and sideslip probes is such that single fail operation corrected angle of attack and sideslip is able to be derived. The autopilot sensors (e.g. attitude, height, heading) are simplex and interface with FCC4 only. They enable a limited authority, fail safe autopilot facility to be provided.

System and Computer Architecture

The four flight control computers (FCC) are operated with software controlled, synchronous program cycles, and input sensor data is interchanged by optically isolated serial data links, as illustrated in Figure 6.

This synchronisation and consolidation enables a bit for bit identical control law implementation to be carried out in each computer.

The in-flight monitoring concept is threefold:-

(1) Software comparison of interlane data to implement the required voter monitor functions.

(2) Independent checks of critical computer functions by a dedicated hardware monitor. This is supplemented by calls to a sub-set of the resident self test software modules. This sequence of calls is completed every 42 seconds.

(3) The use of a duo-triplex primary actuator to provide absorption of actuator failures or actuator input defects.

Based on the necessity of having a continuously available primary flight control system together with the decision to utilise synchronised voter/monitors to ease the redundancy management task the design evolved into a multiple similar digital solution.

The flight control computer itself was designed to ease the integrity assessment task as follows:-

(1) The machine instruction set was identical to that successfully used in previous programmes, eg, Boeing YC-14 AMST. This instruction set was also limited in size to ease the microprogram and software analysis task.

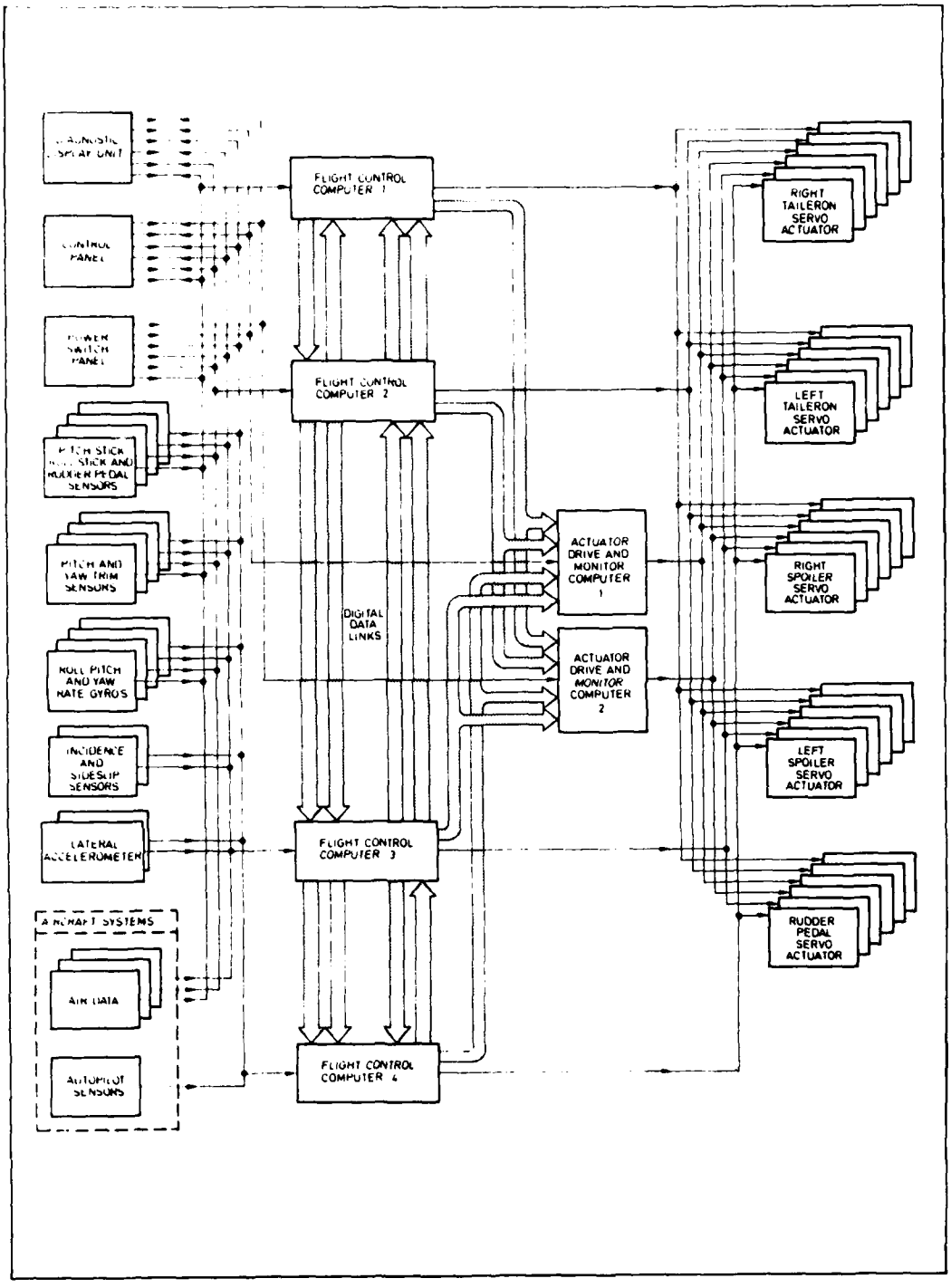


Figure 5 Flight Control System Configuration

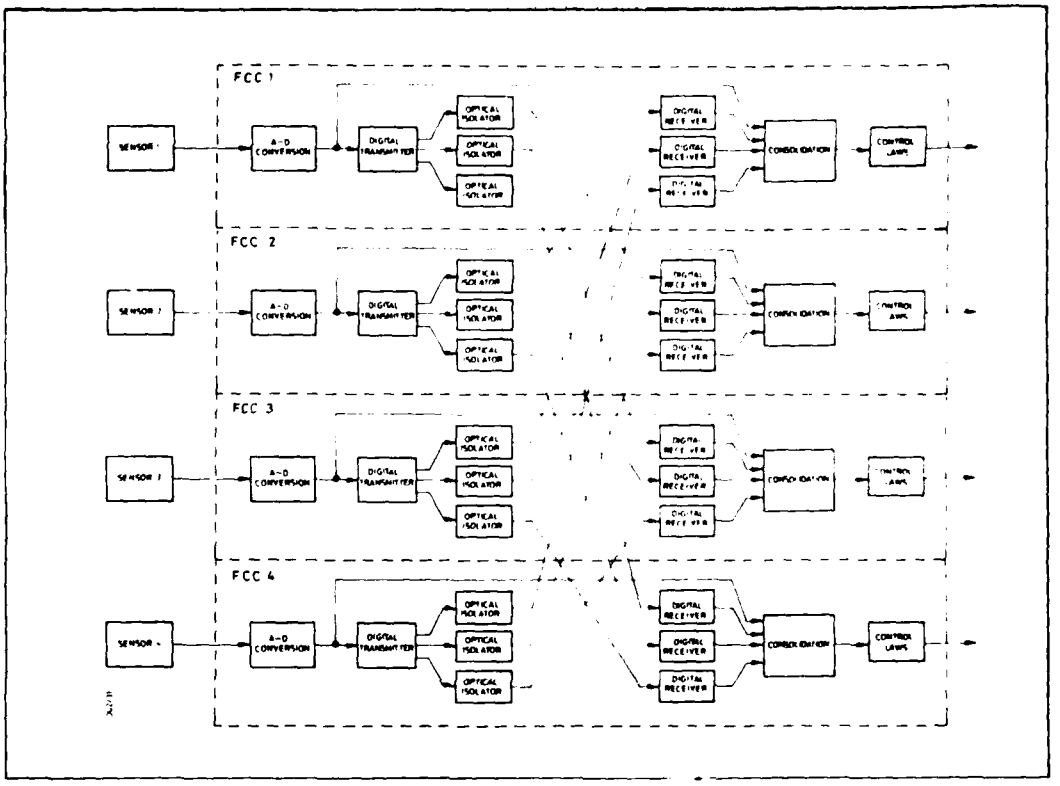


Figure 6 Consolidation of Quadruplex Input Data

(2) The central processor unit (CPU) was designed by GEC Avionics to provide maximum visibility and limit reliance on device manufacturers failure data.

(3) The software support tools were derivatives of those previously used on a number of programmes and were of proven maturity.

Software Architecture

The real time control is achieved by a hardware master reset timer which calls a non interruptable executive. The executive then calls the frames (processing time slices containing related functional modules) in a defined sequence to provide the required iteration rates for the various computing paths. Each frame typically contains control laws, with related signal selection and logic module functions, and consists of a set of program modules each defining a function that is easily defined, implemented, tested and audited. The worst case run time of a frame is controlled at the design stage to ensure that the computing task is completed before the master reset occurs. Should any fault occur that causes the frame run time to exceed the master reset time interval, this is detected and flagged as a computer fault.

The structure of the flight resident program is shown schematically in Figure 7.

The software design as with the A310 system, followed a formal heavily structured procedure. The basic difference was that the software suite would be effectively unmonitored except for normal frame completion checks.

The development process, illustrated in Figure 8, shows the extensive review, module testing and integration procedures implemented to ensure the software suite accuracy.

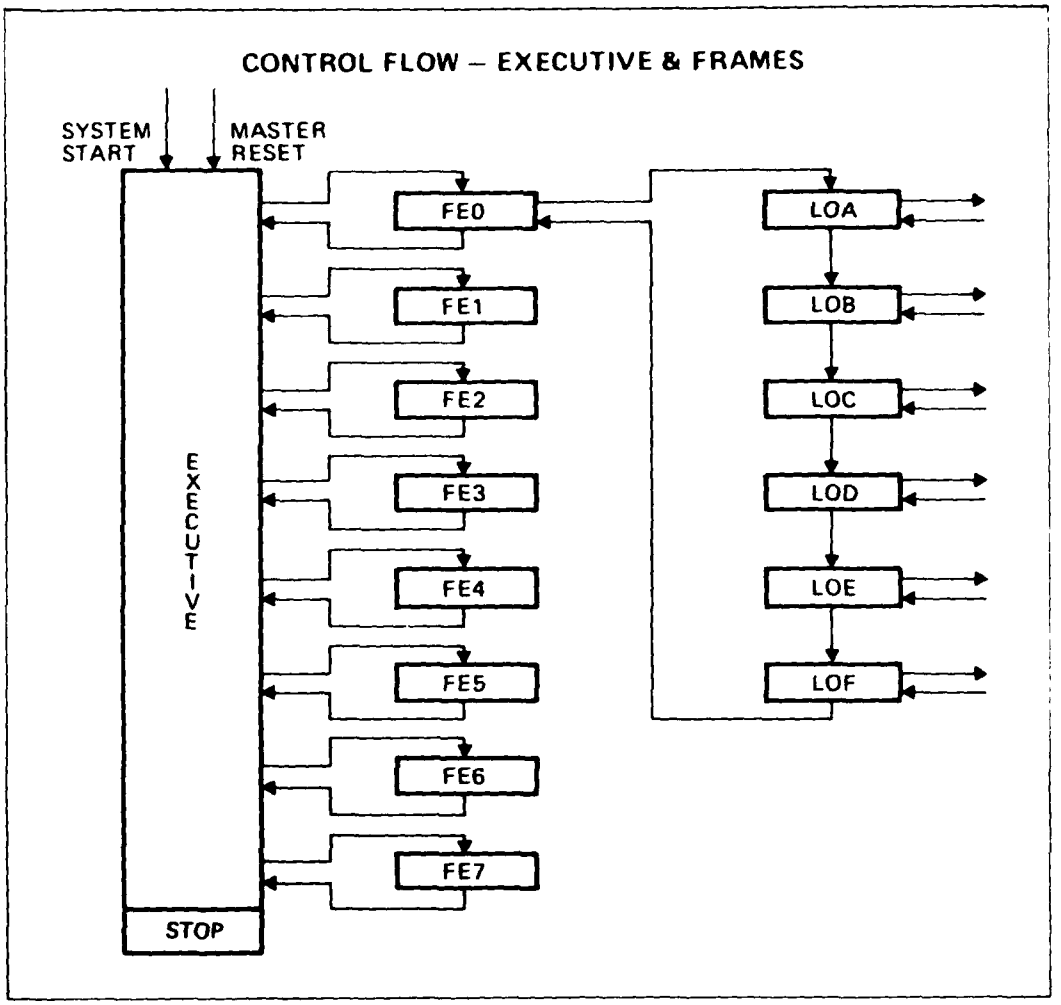


Figure 7 Flight Resident Program Structure

A310 SLAT AND FLAP DEVELOPMENT EXPERIENCE

The computer design and development programme began with contract award in November 1979 and first aircraft deliveries were achieved to airlines in April 1983.

During this time the hardware concept remained unchanged with the only major hardware expansion, in the interface areas, occurring within the initial six month period.

The first flight standard hardware was delivered, on schedule, in September 1981, to support the flight test programme which began with the first flight in April 1982.

Initial design of the unit included one spare module allocated to each channel of five modules. No encroachment into this area has been required during the programme.

During the course of the development programme thirteen software suites were formally issued, some issues being the result of only minor constant or logic changes affecting only one of the two lanes.

As can be seen from Figure 9, the final in-service software suite is four times larger than the initial estimates. This increase is due to two main factors:-

* Under-estimate of task, including the effects of extremes in the operating environment such as hydraulic pressure fluctuations.

* Increased requirements both as new tasks and from specification refinement.

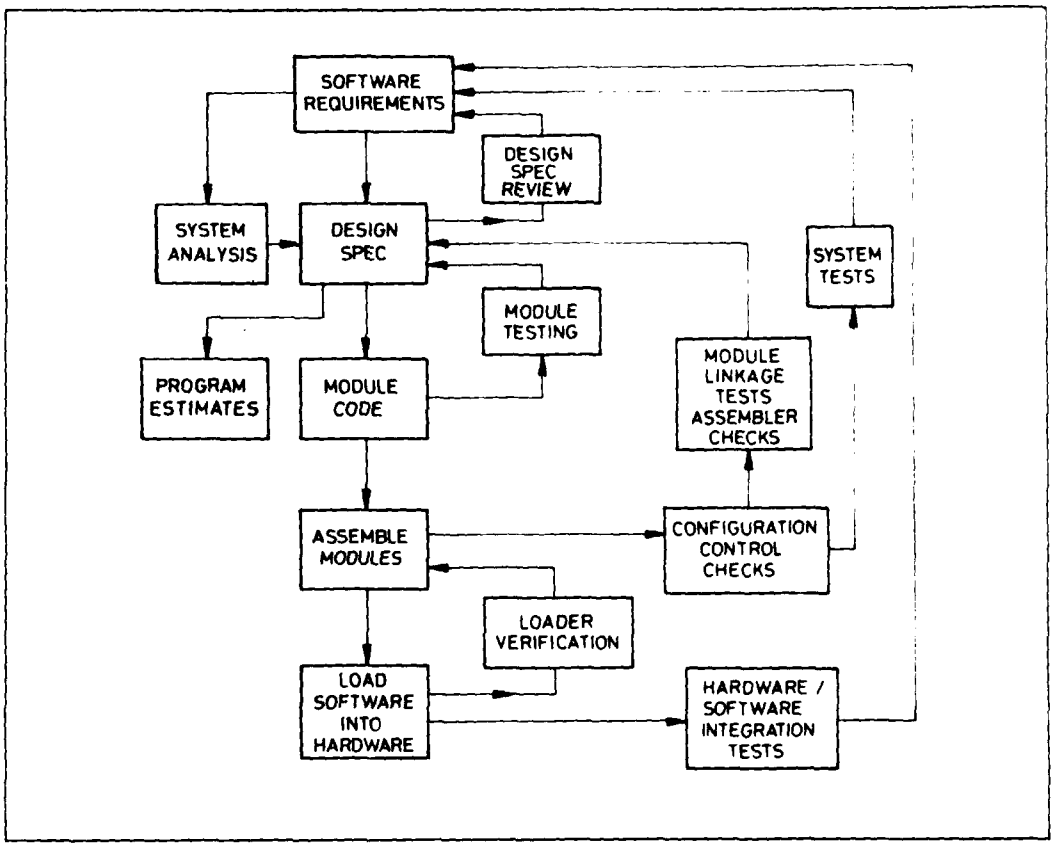


Figure 8 Software Development Process

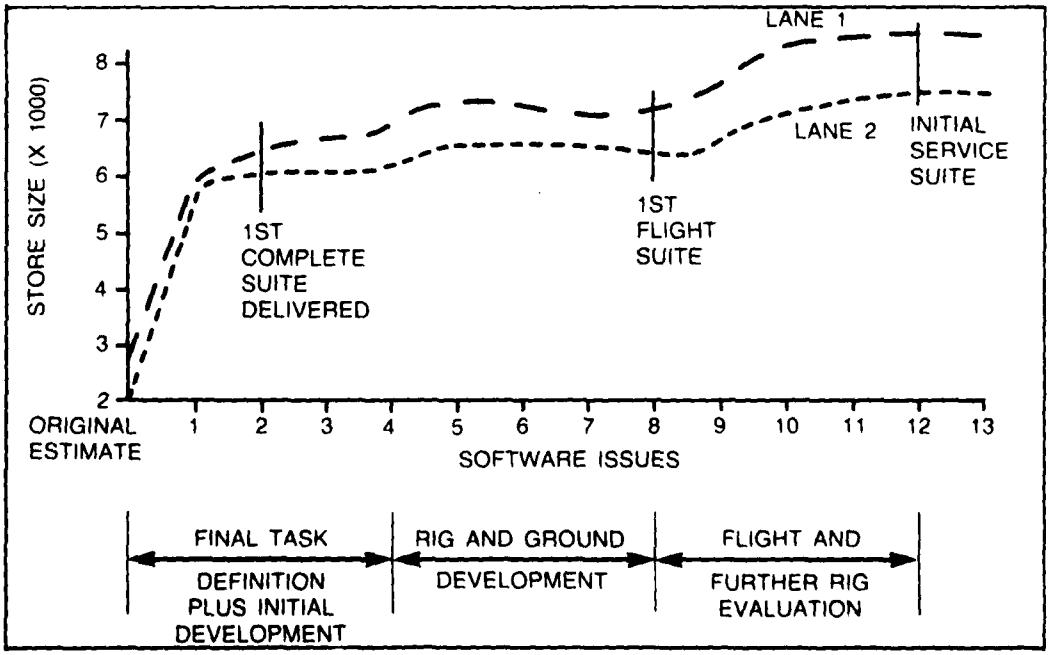


Figure 9 Growth of Software

* The difficulty in translation of a well-known mechanical function into terms compatible with digital implementation.

This growth in software is not a new phenomenon in the industry nor was it unexpected in the computer design. It has been accommodated within the designed growth margin of the computer, but goes to emphasise the critical importance of including adequate growth margins both in store and runtime in new developments.

In addition the inherent flexibility of digital computers, including those such as the AJ10 Slat and Flap Control computer with embedded software, has been shown. Changes in output interface operation and major functional additions have been accommodated during development without modification to the computer hardware other than replacement (E)PROM devices.

The cumulative totals shown in Figure 10 for change requests include all those necessary to support documentation alterations and the continual production improvement programme associated with new equipment.

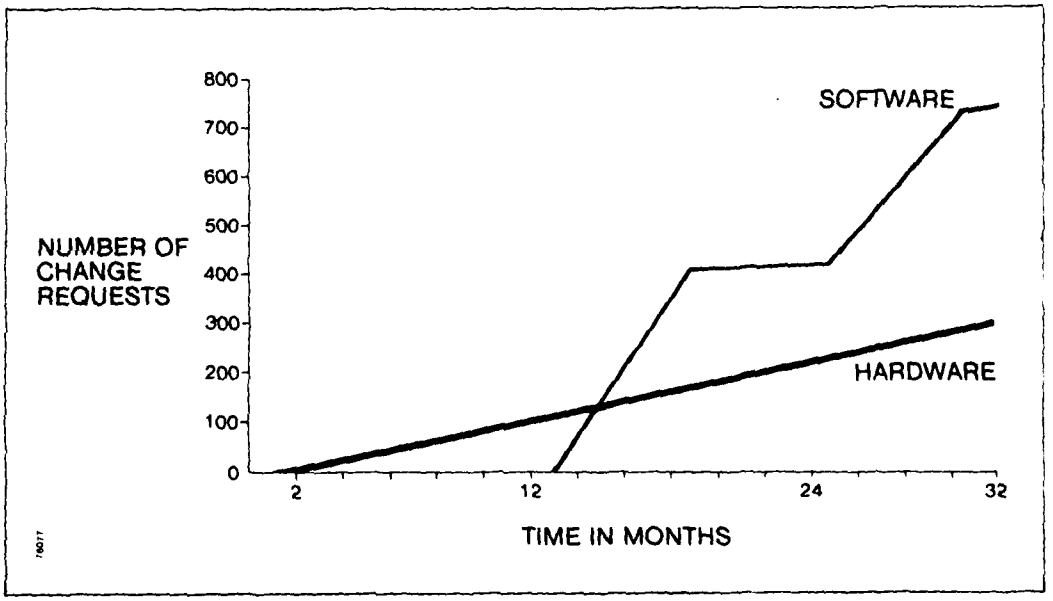


Figure 10 Changes (a) Software
(b) Hardware

One early joint decision among the participating companies was to standardise on the microprocessor host system to be used for development. This commonality enables easy transfer of data between groups and made possible 'on the spot' investigations of problem areas using emulation facilities.

Care was, however, taken throughout the development process to ensure that dissimilarity of host systems was retained during production of formal software suites for flight purposes.

JAGUAR IFCS DEVELOPMENT EXPERIENCE

The design and development activity formally started in 1977 with the award of a contract to British Aerospace by the Ministry of Defence. This award was the outcome of a Ministry Of Defence-funded study programme. The implementation utilised experience gained by British Aerospace, Dowty Boulton Paul and GEC Avionics throughout the early 1970's on previous aircaft such as the YC-14 AMST.

As with the Airbus programme, once again the original hardware concept remained the same, with ground based rig dedicated equipment being delivered in 1978.

Following extensive rig and aircraft ground trials, including a considerable amount of electromagnetic compatibility (EMC) and power supply transient testing, the first flight took place on 20th October 1981. Flight testing of the fixed gain control laws was completed in 13 flights, compared with the 14-22 flights budgetted. The aircraft proved easy and straightforward to fly with excellent FCS reliability. The flying rate of the aircraft was never limited by any problems within the FCS but solely by the large amounts of data to be analysed between each flight.

During this 4 month period only one FCS LRU was exchanged due to a defect. The LRU change was prompted, during routine servicing, by BIT detection of a spurious cross lane data transmission malfunction. No in-flight computing malfunctions occurred throughout these trials.

The Jaguar IFCS programme produced four formal flight cleared software suites plus a number of interim standards for rig and ground test evaluation.

The four suites actually flown were designed as part of the continuous flight envelope extension programme as follows:-

1) First flight standard providing fixed gain control laws for a stable aircraft.

2) Software update providing enhanced BIT and scheduled gain control laws for a stable aircraft.

3) Initial fixed gain suite for early flights with a $-3\%\overline{C}$ to $-5\%\overline{C}$ instability.

4) Complete gain scheduled suite for flight use up to $-10\%\overline{C}$.

The first program store estimate of 8k words grew to 15k words of 16 bits comprised of approximately 400 modules for the initial flight standard.

The development time scale and the number of change requests raised during the design, development, analysis and the extensive rig proving phases is shown in Figure 11. The largest proportion of changes were generated by the detailed module testing and analysis conducted in the first half of 1979 prior to the first complete assembly/linking in July of that year.

Further peaks in October 1979 and May 1980 occur when new test conditions were invoked, for example first systems rig exposure in May 1980.

A total of 1300 change requests were raised during this period, of which over 50% were due to requirement errors or misinterpretation.

Immediately following release of the initial issue of Flight Resident Software (FRS), a revision was commenced to incorporate scheduled gain control laws, to enhance the BIT function and to rectify problems encountered during the early trials which had not necessitated immediate correction. This proved to be a very extensive modification exercise resulting in changes to some 75% of the 400 modules comprising the FRS. However the timescale and cost of preparing the new issue was very much less than for the initial issue, and by building on the system integrity appraisal techniques established for the previous system standard, the clearance was achieved with less than 20% of the effort required previously. The major changes in system performance required were achieved with only the single hardware modification which changed the contents of the programme store devices.

Recognising the problems of cost, timescale and integrity associated with software modifications, additional software segregation was introduced to the FRS at this issue. The 21K words of software required were partitioned across 26K words of store. This was organised not only to provide software segregation at module and segment level, but also to contain different sections of software within separate programme store devices. The objective was to enable future software changes to be contained to a minimum number of software modules and programme store devices. Thus bit for bit comparison of successive FRS assemblies would easily identify the change areas and enable subsequent verification and validation to be more localised than could be justified if the new assembly changed all of the programme store instruction locations.

Integrity Appraisal

The specific requirements of a full time primary fly-by-wire system forced GEC Avionics and our customer British Aerospace (BAe) to institute a major activity to appraise the IFCS integrity, especially the common mode software area.

The integrity of the IFCS is primarily determined by the system architecture. Therefore the elements of maximum concern are the points where the redundant lanes are consolidated or otherwise connected, together with the potential for common mode safety critical design defects in the hardware, firmware or software.

The appraisal was carried out using both 'bottom up' and 'top down' analyses, and since some of the issues involved could not lead to useable quantitative estimates of risk, qualitative assessments were also necessary.

The main elements and interactions of the appraisal/audit include:-

i) 100% coverage single fault Failure Modes and Effects Analysis (FMEA).
ii) Multiple fault FMEA for specific combinations.
ii) Flight resident software audit.
iv) Appraisal of special areas.
v) Configuration inspection.
vi) Qualification programme.
vii) Burn-in programme.

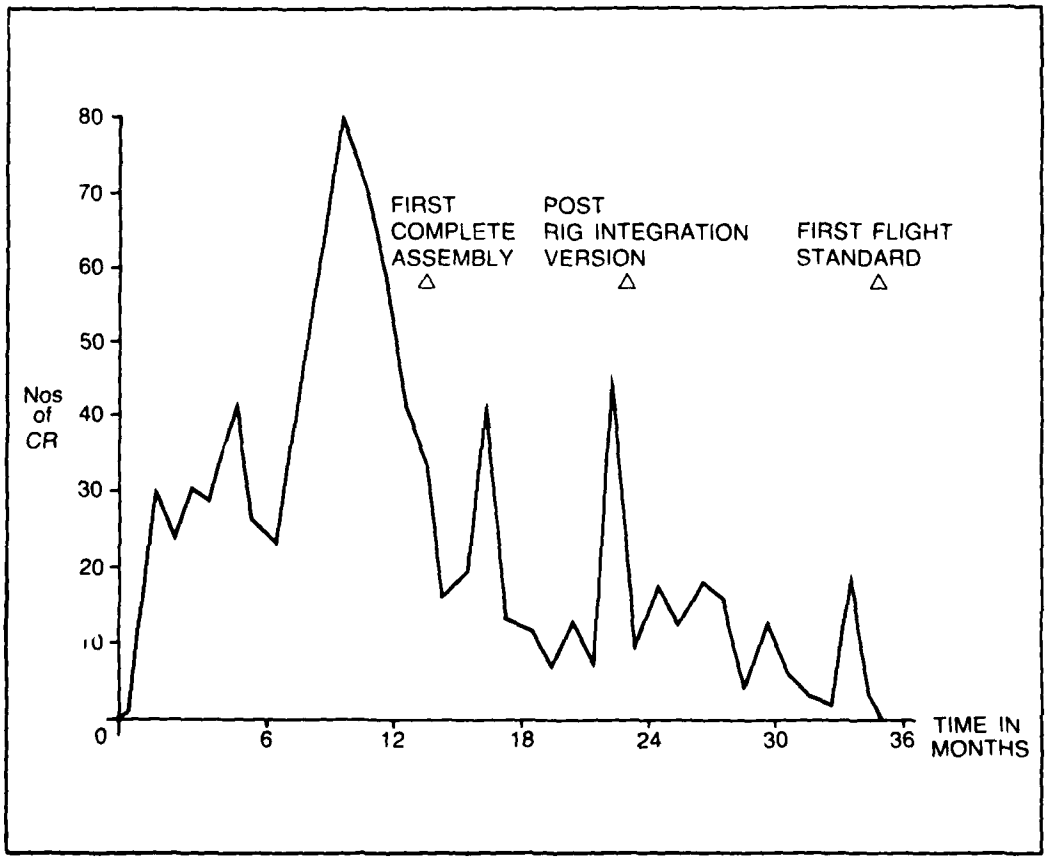


Figure 11 Jaguar IFCS 1st Software Issue Development Programme

These primary elements were supported by

a) Module, chassis and unit FMEA.
b) Microprogram appraisals.
c) Voter/monitor appraisals.
d) Tolerance analyses.
e) BITE coverage analyses.
f) System architecture analyses.
g) Reliability analyses.

During the course of the appraisal detailed technical evaluations of various features and functions of the IFCS were made. The requirements for these evaluations were generated mainly from the FMEA activity, and by BAe as a result of their test activities.

These evaluations were reported as a series of Technical Appraisals appended to the overall integrity report, and their results incorporated into the risk assessment.

The integrity appraisal was conducted by a team with specialist knowledge of the equipment design, but to ensure rigour in the appraisal they reported to an independent authority consisting of senior engineers from GEC Avionics and BAe.

An essential part of the system clearance depended on the extensive emulator, rig and aircraft testing carried out at BAe, Warton. During these exercises, any unexpected observation that could not immediately be explained by the personnel involved in the test resulted in the raising of a formal query. A written response to every query, approved by both BAe and GEC Avionics, was a mandatory requirement for final Quality Assurance clearance of the aircraft for flight.

MAINTENANCE AND RELIABILITY

Specific attention was placed on these two aspects during the design phase of both these FBW programmes in order to reduce the final operators cost of ownership. This section will firstly discuss the concepts used in both designs to achieve this aim in an optimum manner and then describe the in-service experience to date of the A310 system.

Design for Maintenance

The maintenance philosophy chosen splits into two areas:-

Reporting by the computer on the status of peripheral system components, eg, electro-hydraulic brakes, hydraulic motors, air data sensors.

Comprehensive self test capability together with a functional modularised design.

System Reporting - One major advantage of integrating the system protection features into the controlling computer is the facility of providing information on the status of critical elements, eg, electro-hydraulic brakes as a replacement for scheduled mechanical inspections/checks on previous generation civil secondary flight control systems.

Such mechanical protection systems could require partial dismantling of torque shaft runs to check the 'blow back' protection features.

The A310 Slat and Flap control computers for example, continuously monitor and report, by non-volatile display, the status of the following peripherals:-

* Hydraulic Solenoid Valves
* Hydraulic Pressure Switches
* Pilot Control Lever Interface
* Synchro Position Pick Offs
* Wing Tip Brake Coils and Power Supply
* Air Data Sources
* Flap Vane Position and Jam Switches
* Kruger Flap Position Switches
* Hydraulic Motor Performance (by Speed Monitoring)

Self Test and Fault Diagnosis - As with all digital computers a comprehensive BIT facility is provided for a minimal increase in complexity. The BIT system for the A310 Slats and Flaps control computer provides, for example, coverage of 79 percent of the unit total failure rate.

Note - This is a pessimistic figure as, for example, all failures of filter and bypass capacitors were considered to be undetected.

The fault coverage is limited mainly by the specification requirement to provide isolated, low leakage current, output discrete interfaces. Extension of BIT coverage across isolation devices was considered uneconomic for the benefit received.

During the design phase of both systems attention was given to functional allocation to modules, so as to ease the diagnostic task.

Reliability

As with all avionics, the aim is to maximise the unit reliability. The initial design aim at A310 contract award, for example, was to achieve a mature computer MTBF of at least 10,000 hours.

To achieve this aim the following principles were adopted:-

* Preference was given to low dissipation circuit designs, where performance allowed.
* Conservative component derating figures were applied during circuit design and verified at design reviews.
* High failure rate components are specified as high reliability burnt in parts from suppliers.
* All completed units are subjected to a burn-in procedure prior to delivery to remove component infant mortalities.

Current Experience with the A310 System

Although the use of Fly-by-Wire techniques within a Slat and Flap control system is a new development for airline use, operator acceptance has been high.

At this time, the Airbus Industrie A310 is in revenue service with 15 airlines and the feedback available confirms our original MTBF prediction of >10,000 hours by achieving an actual MTBF of 12,938 hours. The current ratio of MTBUR to MTBF is two-to-one which we expect to decrease with increasing operator experience. Regrettably, advances in this area can easily be nullified by the ease of removal of digital computers when compared to mechanical units.

GEC AVIONICS EXPERIENCE WITH THE USE OF DISSIMILARITY

As previously mentioned the technique employed for passivating software errors with the A310 Slat and Flap control system is dissimilarity from the design requirement level down.

The monitoring scheme is to constantly cross compare the safety critical outputs computed by each lane and to diagnose a computer fault for a mismatch lasting longer than a preset interval. This can be likened to a real time test against a simulation throughout the operational life of the unit.

This simplistic technique has the disadvantage of trading availability for integrity, which although acceptable in this particular application, can not necessarily be extended to other flight control applications. Investigations into more sophisticated dissimilarity techniques are underway and are outlined in the section "Future Development in Dissimilarity".

The basic technique has, however, been proved successful, as from the initial system rig evaluation, through flight test, into airline service no uncommanded surface movement has occurred. All software errors which have had the potential to cause such a hazard have resulted in computer fault indications with resultant channel shutdown.

Dissimilarity also has not, as could be imagined, involved GEC Avionics in a doubling of the software non-recurring cost as the testing/integration technique used avoided the extensive and exhaustive detailed software FMEA and module testing required by multiple similar systems.

The testing philosophy of using the dissimilar lanes as 'test harnesses' for each other, has also been successful. The result has been that crucial software errors are discovered during initial integration testing, and are manifested by computer faults. However, some low integrity errors may be masked by the dual nature of the design. This has now led us into separately integrating each individual lane of software with the hardware, using a dummy routine in the other lane before complete hardware/software integration occurs.

Overall, we consider our initial A310 decision to use dissimilarity to have been validated and have carried this concept forward into the A320 Slats and Flaps system solution currently in development. This A320 system will also include the use of High Order Languages as part of the implementation.

Figure 12 shows the cumulative number of discovered software errors against time, annotated by software issue and event.

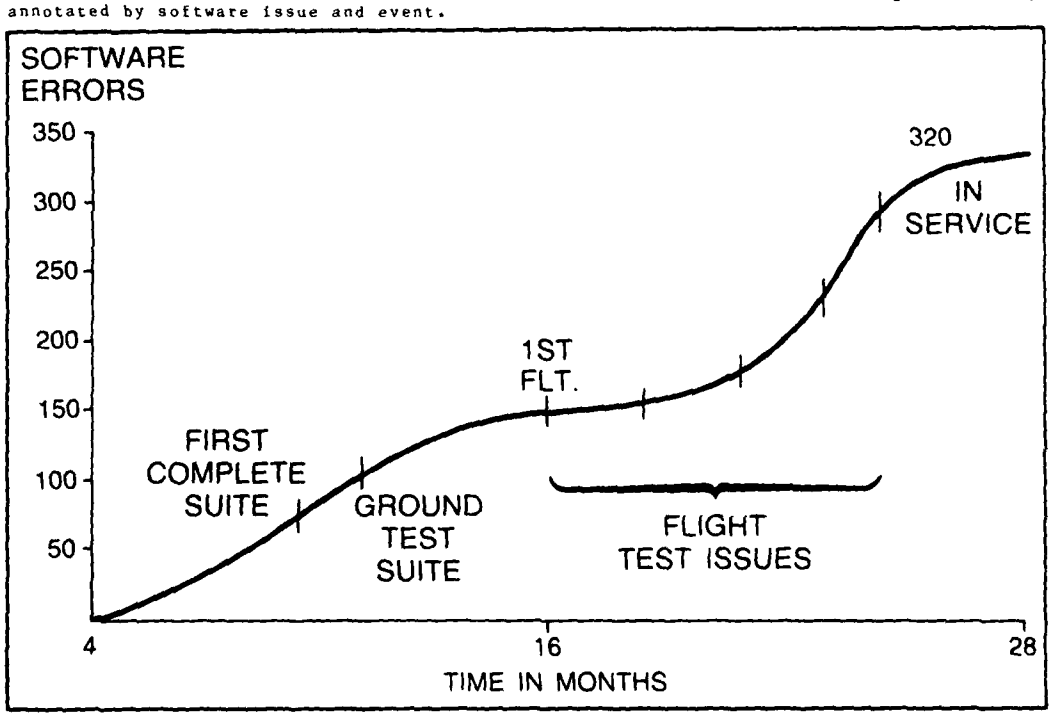


Figure 12 Software Errors Found - Cumulative Total

As can be seen from Figure 12, the flight resident program was mature at one standard prior to first flight. Then, extended testing at corners of the operating envelope, eg, fluctuating low hydraulic pressure associated with repeated and rapid changes in requested surface position, has resulted in additional changes being incorporated mostly prior to airline delivery with the remainder being swept up in an in-service software update by service bulletin.

FUTURE DEVELOPMENTS IN DISSIMILIARITY

During the course of the application of dissimilar techniques to the A310 Slat and Flap system, a number of areas became highlighted as worthy of more attention, as follows:-

* Ensuring dissimilarity.

* Risk of multiple channel disconnects for one residual software error.

* Lack of capability to provide a failure operational system for a residual software error.

* Cost penalty involved in supporting independent multiple software teams.

To address these problem areas, an alternative approach, first proposed by Professor J Shepherd, is being studied by GEC Avionics together with the Cranfield Institute of Technology under contract from the Royal Aircraft Establishment (4). Essentially, the method follows the known concept of error recovery blocks but evaluates the alternative block concurrently with the primary block.

The method depends upon two techniques:-

* The concept of temporal separation of software channels.

* The concept of deriving an alternate version of the program from the primary version.

These techniques are discussed below.

Temporal Separation

The concept of temporal separation is essentially based upon the fact that while spatial separation prevents the proliferation of hardware faults, software faults are time dependent. Thus, to ensure that a software fault does not affect all channels simultaneously, it is necessary to ensure that each channel does not process identical data. The method of ensuring this depends upon the fact that for a real time system such as flight control a high iteration rate is used. The method is then to use inputs from different time periods for each of the channels.

Having separated these channels in time, it becomes possible to validate both the hardware and software operation before committing that iteration to control of the aircraft.

This validation is achieved by comparing the computed outputs/functions of one program with that of an alternate version, using the identical data. This technique also enables hardware faults and software errors to be isolated.

Having then isolated a suspect software channel, it is suggested that a Langrangian Extrapolation of the previously validated outputs is used to control the aircraft until the alternate program versions are found to track once more within tolerance.

For this method to function it is of course necessary to construct an alternate version of the program. The method proposed to do this is described below.

A Method of Generating an Alternate Version of a Program

In considering software monitoring, it is necessary to consider the four types of operation encountered in a program. These are:-

* Arithmetic Operations
* Logical Operations
* Decision and Branch Operations
* Input/Output Operations (Including Interrupts)

Each of these types of operation are considered in turn.

Arithmetic Operations - As an example, to produce an alternative implementation of an arithmetic operation, difference equations could be formed to allow computation of a variable, based on the change since it's previous computation.

Logical Operations - In addition to arithmetic operations it is necessary to consider logical operations of the form:-

A : = B + C
A : = B . C

Where:-

+ = OR
. = AND

In general with these types of equations, it is easy to produce an alternative algorithm, based on De Morgan's theorem.

In general, as is common in logic design any Boolean function can be expressed in terms of either NOR or NAND operations and thus, there are always two alternative expressions for logical operations. Thus, the requirements for having different algorithms for the main and check programs are satisfied.

Decision and Branch Operations - These operations are typically of the form:-

```
BEGIN

IF X = Y THEN BEGIN
         MODULE A
         END

         ELSE BEGIN
         MODULE B
         END

END
```

It will be seen that this type of operation is a combination of arithmetic and logical operations and can be treated as such.

Results of Preliminary Study

In order to check the validity of the approach, software representing the short period mode of an aircraft was developed and the techniques described above were applied. Approximately 60,000 random faults were introduced for each of three different input signals (which had random noise superimposed to ensure that realistic signals were used).

The results obtained were:-

	Sine Test	Triangle Test	Sawtooth Test
Number Of faults	61127	62019	61063
% Detected	100	100	100
% Errors Corrected	83.9	82	75.6
Acceptable Errors (1 - 15%)	16.0	17.9	24.3
Unacceptable Errors	0.1	0.1	0.1

In practice the sinusoidal input is more nearly representative of the type of input used in flight control.

It should be realised that no attempt was made to optimise iteration rates or tolerance levels during this study and that therefore it is to be expected that even better results can be obtained.

The results obtained, however, indicate that the proposed approach is valid and that it is worth considering the concept further.

CONCLUSIONS

Both the Jaguar IFCS and the A310 Slat and Flap system have been successful FBW implementations of flight control systems. A number of important lessons have been learned and an equal number relearned or emphasised during the course of these successful programmes.

These can best be described under three headings - System, Software and Hardware.

System

The ability of a primary digital fly-by-wire system to provide normal aircraft handling to the crew for unstable air vehicles was proven, the Jaguar program having achieved [illegible].

The crucial importance of having representative rig evaluations conducted early in the programme including exercise of the complete operating environment.

The improved maintenance data available from the system together with the deletion of planned mechanical inspections for integrity reasons points the way forward.

The decision to adopt a dissimilar approach to the A310 system solution in both hardware and software eased the certification burden.

Software

As with all software tasks, the need for a clear unambiguous specification to ensure accurate software system design.

The need for close interaction between airframe constructor and flight control system supplier was highlighted to ensure maximum agreement/understanding at requirement level.

The need within a dissimilar duplex arrangement to perform simplex testing before total integration level to improve availability and remove possible dormant errors.

Related to the first conclusion, ambiguities became highlighted during development by the inherent different interpretations occurring with separate software teams implementing the same requirement.

The common host arrangement between all parties in the A310 development programme enabled problems to be studied on site and in parallel without transportation difficulties.

Both programs have highlighted the need to review the re-validation and verification requirements associated with software, following implementation of modifications.

For high integrity applications, software is particularly critical. All software changes must, therefore, be controlled by the design authority who alone was party to all design and certification decisions.

Hardware

The inherent flexibility of digital systems has resulted in very few hardware modifications in either of the programmes.

The need for adequate growth potential both in store and runtime to be designed in. This growth margin could be as much as 5:1 over original estimates for new tasks for which no previous experience is available.

The reliability of current generation computer hardware, has proven to be consistent with predicted MTBFs of the order of 10,000 hours. This is partially due to the low thermal dissipation devices now available in the market place.

References

(1) Mr. Andrew D. Hills - A310 Slat and Flap Control System Management and Experience (5th DASC, November 1983)

(2) Mr. K.S. Snelling - Certification Experience of the Jaguar Fly-by-Wire Demonstrator Aircraft Integrated Flight Control System (AGARD Conference proceedings No. 347)

(3) Mr. R.E.W. Marshal, K.S. Snelling, J.M. Corney. The Jaguar Fly-by-Wire Demonstrator IFCS (Advanced Flight Controls Symposium - August 1981).

(4) Professor J T Shepherd - 'Some Approaches to the Design of High Integrity Software' (Cardec pp 29 - 38, June 1983).

Acknowledgements

The Jaguar Fly-by-Wire programme has been carried out with the support of the Procurement Executive Ministry of Defence.

The author acknowledges the special team effort that the Jaguar IFCS programme has involved. Success depends upon the combined efforts of British Aerospace, Dowty Boulton Paul and GEC Avionics. In particular the programme would not have commenced or been able to continue without the enthusiastic support of the British Ministry of Defence and RAE Farnborough. His thanks are extended to colleagues in British Aerospace and the Combat Aircraft Controls Division of GEC Avionics for assistance provided in the preparation of this paper.

The author also wishes to acknowledge the valuable contribution of Liebherr-Aero-Technik GmbH to the success of the A310 and its derivative Slat and Flap control systems.

REDUNDANCY MANAGEMENT OF SYNCHRONOUS AND ASYNCHRONOUS SYSTEMS

by
Gregory M. Papadopoulos
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

SUMMARY

While asynchronous systems may initially appear attractive due to the "uncoupling" of the channels, the cross-channel interactions are much greater than what might be expected. Both synchronous and asynchronous systems share the burden of cross-channel *consistency maintenance*, the requirement that inputs and internal states are not allowed to diverge too far from each other. In fact, consistency maintenance often dominates the engineering process in a correctly designed system. In asynchronous systems, consistency maintenance takes the form of cross-channel *equalization* along with techniques for handling discrete changes in operating mode. In synchronous systems, consistency maintenance is implemented with *source congruence* algorithms.

Synchronous and asynchronous systems must both support reliable *resolution* of redundant channel outputs, the process of isolating correct effectuator commands from faulty ones. A correctly designed synchronous system may rely on exact bit-for-bit voting to isolate faulty channels, while an asynchronous system must employ more heuristic means such as threshold and reasonableness tests.

We conclude that data synchronous principle is the approach of choice. These systems provide general purpose vehicles on which increasingly complex and diverse flight critical applications may execute.

1. INTRODUCTION

The nature of flight critical functions is evolving rapidly. As automation is aggressively applied, more and more functions are being absorbed into the flight critical realm resulting in a basic conflict. While automation may dramatically enhance an aircraft's effectiveness and survivability it simultaneously increases the risk and consequences of equipment malfunction. For instance, automated terrain following may significantly improve the chances of survival by reducing exposure in hostile environments but a failure of the terrain following mechanism could have catastrophic results.

Of course the same observation can be applied to traditional fly-by-wire control. The performance and efficiencies of open loop unstable airframes must be balanced against the costs of highly reliable control systems. But there is an important difference. It has to do with the nature and complexity of the functions involved.

We claim that the *redundancy management developed for traditional flight control systems does not necessarily generalize to more complex functions.* In fact, the entire class of data asynchronous redundant computers may fail to effectively support the increasing demands on flight critical systems.

It is the goal of this paper and its sequel [1] to methodically develop the concept of a general purpose, fault tolerant computing system on which a great variety of algorithms may successfully execute. Importantly, we insist that the applications programmer be insulated from the low level details of the implementation. Applications ought to be written assuming that they will execute on a *single*, very reliable computer. An application algorithm is developed to specify the control laws, sensor and actuator fault detection, isolation and recovery mechanisms (FDIR), and adaptations to changes in the plant's behavior. The role of the computer *redundancy management*, therefore, is to ensure the application continues to run correctly on a *redundant* machine in the presence of a specified number of malfunctions. We endeavor to separate the problem of writing correct programs from the problem of providing a correctly functioning vehicle on which to execute these programs.

Unfortunately, few systems can actually guarantee correct operation in the presence of even a single arbitrary fault, let alone provide generality, extensibility, and application independence. Systems requiring exceptional reliability cannot be reasoned about heuristically or informally. It is impossible to empirically determine that a system meets a very high reliability goal. It is only possible, by negative experience, to deduce that it does not meet expectations.

There are, however, basic theoretical and practical results in the area of redundant computation, specifically the problems consistency maintenance [2][3][4], and synchronization [5][6][7]. We will attempt to translate these theories into a model that can evaluate the effect of engineering decisions upon the ultimate correctness of the system. One such very important engineering decision occurs early in the design cycle and is central in the following discussion,

> ***Should the redundant channels of the system be synchronized (instruction or frame) or allow to run asynchronously?***

We shall explore the implications of this decision in terms of the redundancy management requirements on the system.

2. DATA SYNCHRONY

Initially, digital computers were employed as cost-effective replacements for analog systems and specialized digital logic. In these roles the computer's job was to *emulate* the function it was replacing. Analog inputs were sampled, operated upon by difference equations, and then converted back to analog form.

In these approaches, the concept of an input can only be loosely specified as the "most recent value" or the current sample time. As shown in Figure 1, this may be contrasted with a more rigorous notion of a function operating on a *stream* of input data. The actual data values processed under the sampled formulation is a function of when the program reads a location containing the latest sampled point. In the case of a stream, the formulation is *time independent*. Conceptually at least, the process consumes the next value from the stream when it becomes available. In a real time system we must of course ensure that our process can consume values at least as fast as they can be produced.

We term systems that adopt the sampled data model *data asynchronous*. Systems that enforce the more rigorous concept of data streams shall be termed *data synchronous*. In the case of a single processor the differences between these two approaches are somewhat immaterial. The importance of one formulation over the other becomes very clear in the case of multiple processors, however.

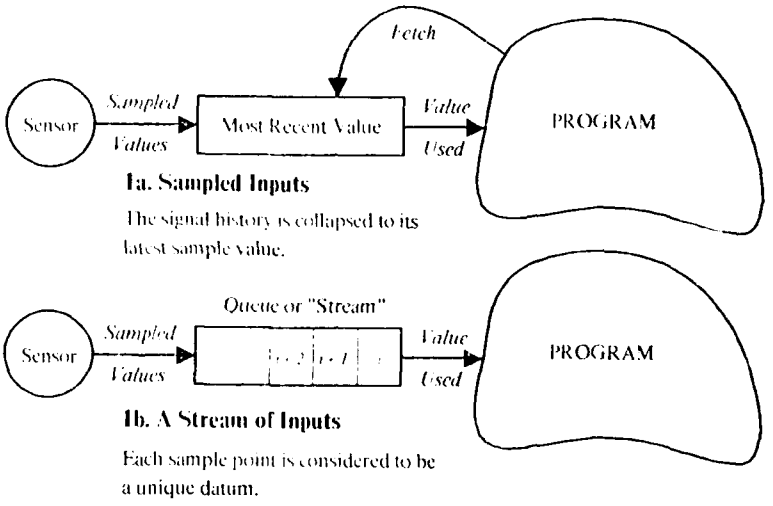


Figure 1. Sampled Inputs Versus a Stream of Inputs.

2.1. Consistency and Synchronisation

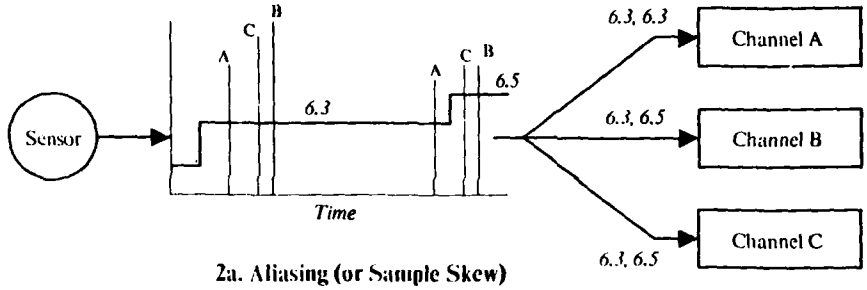
Now suppose that we wish this program to execute on three independent processors and would like to compare outputs in order to isolate processor faults. Given that the programs and processors are deterministic, † the outputs will exactly agree when (1) all processors are fault-free, and (2) all processors receive exactly the same inputs. It does not follow that a faulty processor *necessarily* will produce inconsistent outputs (this is the problem of *latent faults*, dealt with in the sequel), only that inconsistent outputs imply a faulty processor only if the inputs are identical.

This simple yet key idea of *input consistency* is central to correctly operating fault tolerant systems. It is one of the most difficult properties to maintain in a fault-prone environment. While it may appear obvious or even trivial, it is rarely, if ever, performed correctly in production systems. As shown in Figure 2, inconsistent inputs can be introduced in two fundamental ways,

1. *Aliasing*. This is a property of data asynchronous systems. The different processors may sample the inputs at slightly different times. The effects of this input inconsistency on the outputs will be dealt with shortly.
2. *Faulty Sources*. This problem plagues data synchronous systems as well. A faulty source may deliver different values to each of the processors. In this case, a single input may induce the processors to all diverge; a clear, and catastrophic, violation of our ability to identify faulty processors by comparing their outputs.

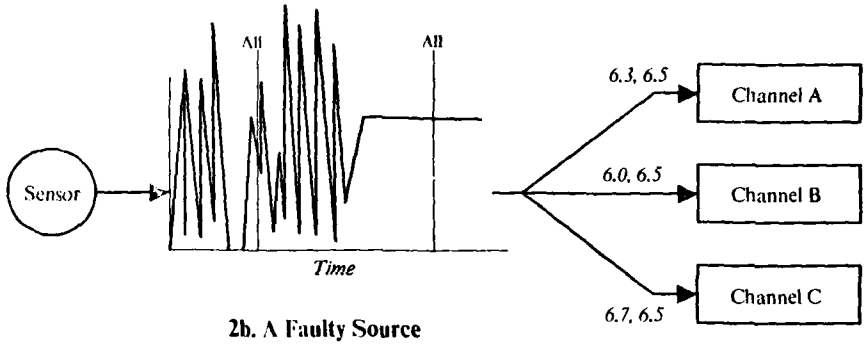
Within our terminology, the problem of aliasing is one of data synchronisation. The problem faulty sources is one of input consistency. Solutions to these problems invariably take the form of the exchange of information between processors. These exchange mechanisms cannot be treated informally. For any dependence created between processors can become a potential path for propagating faults and thus undermine the reason for introducing exchanges in the first place: to prevent a faulty unit from inducing a healthy unit to malfunction.

† To be precise, it is required that the program be a *continuous* and *monotone* function of the histories of its inputs. More simply, continuity requires that a program produce outputs without requiring an unbounded amount of input, while monotonicity prohibits a program from producing outputs before it has received inputs. A fault-free deterministic processor, by definition, must always yield the same outputs whenever executing the same program on the same inputs.



2a. Aliasing (or Sample Skew)

The production and consumption of sensor values are not synchronized. The channels then produce different results.



2b. A Faulty Source

Even with perfect synchronization, the sensor can induce different values into the channels

Figure 2. Aliasing and Failed Sources Can Induce Inconsistent Outputs.

3. APPROXIMATE AGREEMENT IN ASYNCHRONOUS SYSTEMS

One possible solution to the input consistency and synchronisation requirements is to relax the condition of exact agreement among redundant outputs and replace it instead with approximate agreement. Does this then translate to a more relaxed approximate agreement constraint among the inputs? Does it make redundancy management easier or more robust?

3.1. Emulation of Analog Systems

The first use of digital computers in flight control was a cost-effective replacement of analog equipment. The traditional "black box" decomposition was maintained: sensor packages, air data, control law, *et cetera*. The signal protocol between these functions was still in the analog world. Analog inputs would be sampled by Analog-to-Digital converters, internally processed, and then output as an analog waveform through a Digital-to-Analog conversion. To first order, the input/output behavior of these boxes remained the same. They simply emulated the functionality of the analog circuits they were displacing.

3.2. Threshold Tests

In the analog world, the computers themselves are not completely deterministic as a wide variety of random noise sources affect the nominal transfer functions of the components. As such it was impossible to rely on exact agreement of

outputs. Instead, outputs were compared within *thresholds*. A signal and/or its derivatives must lie within some normed distance to a majority of redundant signals, otherwise it is considered faulty.

Strictly speaking a *signal* can't be faulty. Some underlying process (hardware or software) has induced the signal to take on anomalous values. Thus a threshold test can only relate to a hardware fault only to the extent that exceeding or not exceeding the threshold can be reasoned to be a fault in the underlying process.

Thresholds are tricky things and very application specific. Too high a threshold can permit bad outputs to go undetected. Too low a threshold causes nuisance or false alarms. Either way, the reliability of the system can be seriously compromised.

The correct threshold gets more and more difficult to determine as the underlying processes become more complex. In the case of sensor and actuator FDIR, models of these processes are mathematically tractable and thresholds with good detection/nuisance properties are readily derivable and can be supported through extensive simulation. For complex analog systems, and especially their digital emulators, the underlying fault processes are impossible to quantify or even enumerate. Thus the thresholds must be set with respect to the expectations of the nominal signal values. So instead of directly looking for signatures of faults, we are instead asking are *the outputs reasonable*?

3.3. Testing Results, Not Hardware

Under these conditions it is impossible to separate the problems of correctly operating units versus a correctly operating system. The setting of a threshold must consider the dynamic behavior of the vehicle, be sensitive the current operating mode, and tolerate noise. These demands lead to complex thresholding functions, often involving filters and time averages of threshold trips in order to diagnose a channel as faulty. Every time the application is even slightly altered the thresholds must be re-examined. This is the antithesis of application independence. As the complexity of the application increases the dependence becomes even more pronounced.

It is impossible to quantify all of the failure modes of a general purpose computer—the generality that permits programming for any problem also admits extremely complex failure processes. Any experienced computer designer can attest to the truly bizarre behavior some bugs can induce. Intuition gained with hardwired control systems may serve poorly in the design of complex redundant computers.

> *Nobody really understands the failure modes of digital computers. To say "that will never happen" to a demonstrated failure mode is dangerous at best. We must be willing to invest in provably correct redundancy management techniques if we wish to enjoy the benefits of general purpose processors.*

Testing the reasonableness of outputs produced by digital processors is not a direct measure of the health of the underlying hardware. In fact, very sophisticated applications, such as Artificial Intelligence (AI) may require as *much or more effort to determine whether a result is reasonable than it took to compute it!*

A coherent policy of redundancy management must base its decisions on more direct information about the actual state of the underlying processes, in our case digital computers and their interconnect.

3.4. Maintaining Reasonable Thresholds is Difficult

In light of the threshold problems, to be effective the nominal output differences between the redundant channels must be kept controlled and as small as possible. In a data asynchronous system this can become a more difficult task than it might first appear.

Oversampling

By definition, the redundant channels in a data asynchronous system have sample clocks that are independent of one another. The skew between the sample clocks causes each channel to obtain slightly different values for each of the inputs. Importantly, this skew induces a certain phase uncertainty in the compensator when the outputs of the redundant channels are averaged or summed. A common solution to this problem is to *oversample* the compensator, running it at a faster sample rate than what would be required in a simplex implementation.

Oversampling has the obvious effect of increasing the processor loading, often times by a factor of two or more. This increase has to be assigned to the overhead of redundancy management, as it does not exist in a simplex system. This burden can be reduced somewhat by employing special low sample rate control laws, but the simplex case would also benefit from this extra engineering.

Equalisation

Modern control systems often employ forward loop integral compensation and high gain proportional compensation. In these cases, the compensator may be unconditionally unstable or quickly saturate when not closed through the airframe dynamics. If such compensators are simply replicated with their outputs averaged the results can be disastrous. Figure 3 gives a simple illustration of the divergence that can happen in this case. So even if the sampling skew is very small and *the inputs are nearly identical*, the outputs can rapidly diverge. This is because the differences are integrated over time. Even a small input skew can result in unbounded output skew. The only solution to this problem seems to be *cross channel equalization*, whereby the states of the redundant compensators are exchanged an factored into the forward loop. This is shown schematically in Figure 4.

Mode Changes

In a multi-mode control system it is important that all channels change modes simultaneously. This is especially true when reverting to a downmode in the case of a sensor or actuation failure. Smith [8] gives the example of a fly-by-wire system where the primary mode is an alpha command mode. Two redundant alpha vanes are provided as a good compromise between the relative cost and difficulty of measuring alpha and the importance of this measurement in the primary mode. If the outputs of the vanes diverge by more than a certain threshold then the channel degrades to a simple

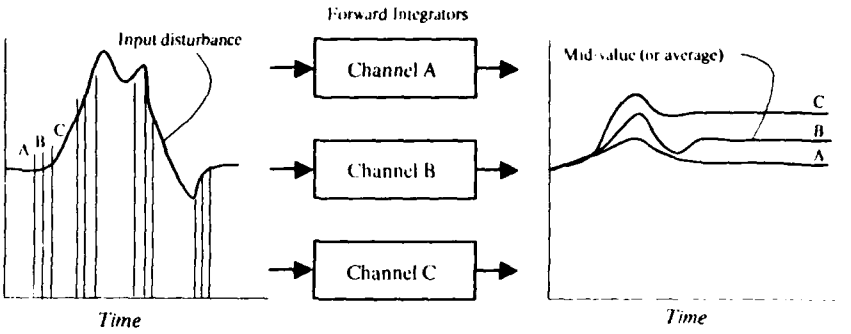


Figure 3. Redundant Outputs Can Diverge Even With Small Input Skew

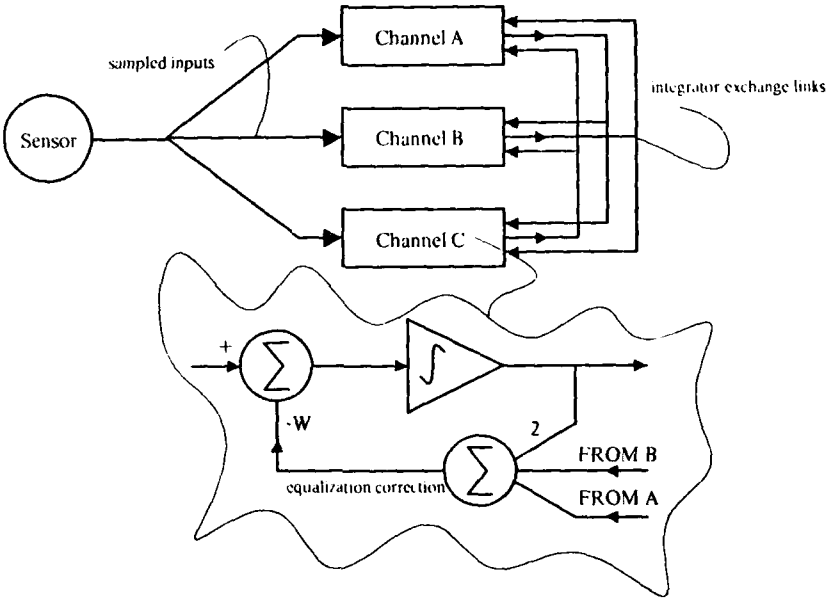


Figure 4. Cross Channel Equalization

stability augmentation system. The divergence of the two vanes might be due to a failure or even a nominal effect such as shadowing during maneuvers.

It is possible in an asynchronous system that when the difference between the vanes is very close to the threshold, some channels might downmode, having perceived the threshold trip, while others may not. If this persisted for any length of time, the channels would certainly disagree on their outputs, causing the obvious erroneous failure conditions.

The solution to these mode change problems is similar to equalization, information must be exchanged among the different processors.

3.5. Cross Channel Dependencies are Risky

The above discussion has illustrated the need for the exchange of information between redundant channels. A good deal of the attractiveness of asynchronous systems, primarily the "independence" of free running channels has been lost. In fact, this very asynchrony can increase the risk and complexity of cross channel exchanges.

For example, the order and relative timing of cross channel exchanges cannot be predicted. This requires the processors to maintain fairly elaborate communications buffers in order to match the similar data from the channel. Some form of time stamping is usually employed to guard against "stale" values or to identify a minor cycle. To guarantee that these exchanges do not provide failure paths from unhealthy channels to healthy ones is indeed difficult.

Consider the case of the alpha mode controller above. Suppose that a maneuver causes the vanes' outputs to diverge very near threshold. As shown in Figure 5, redundant channel *A* perceives a trip, while channel *B* and *C* do not. Normally the exchange of the threshold discretes would resolve this problem. Channel *A* would notice that a majority of the channels did not perceive the trip so it will continue to use the average alpha values and remain in alpha command mode. Now suppose that channel *B* has a failure on the data link that broadcasts its threshold discrete. The failure, quite reasonably, might be such that channel *A* no longer sees the discrete as asserted but channel *C* still does. This will cause *A* to downmode, while *B* and *C* do not. If this continues for a few cycles, *A* will certainly be diagnosed as failed. A *failure of one channel* has induced a failure in another!

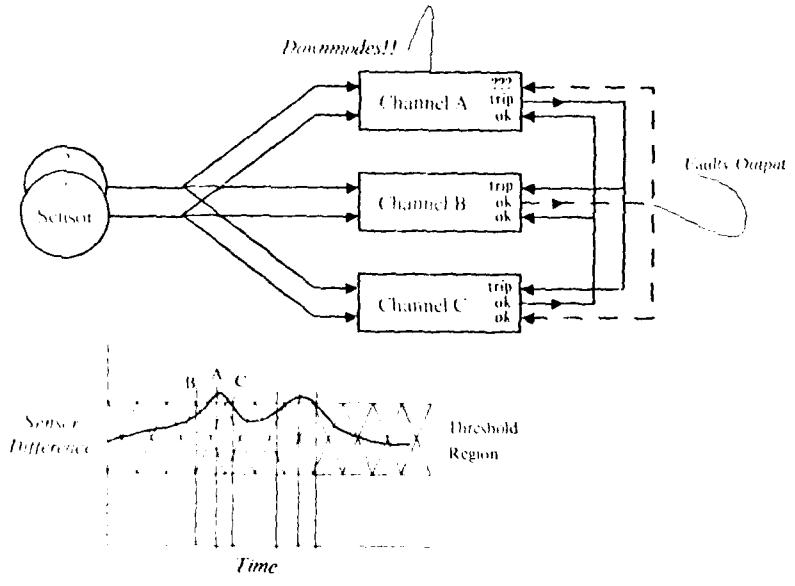


Figure 5. An Inconsistent Exchange of Threshold Discretes.

A whole host of failure scenarios could cause this. A broken wire to channel *A* is a possible cause. So could a marginal or noisy output driver. Channel *C* might be able to recover the signal, while *A* can't. Even heavy encoding doesn't help. Suppose that some form of parity or CRC checks were encoded in the exchange. Channel *A* would, with high probability, notice that *B's* output is bad. What should *A* do? It sees a trip, *C* doesn't, and it doesn't know the state of *B*.

4. APPLICATION INDEPENDENCE THROUGH EXACT AGREEMENT

The problems inherent in data asynchronous systems are traceable to a common condition: *the existence of slightly different views of the system state among the redundant channels.*

Suppose that we invest in mechanisms that, at a very low level, would enforce exact agreement for all data values used by all healthy channels. Given the further assumption that the channels are somehow synchronized to run a common frame rate, the following problems are resolved.

1. *Equalization.* All compensators get precisely the same input values. Because a fault-free digital computer is deterministic, it is guaranteed that all intermediate values and ultimate outputs will be bit-for-bit identical. This is a crucial deduction and provides the basis for application independence.

2. *Mode Changes.* Because all input values will be identical all input FDIR algorithms will yield identical results. Thus all channels will downmode during the same frame cycle. All other discretes will also be consistently distributed, resulting in identical mode changes in all channels.

3. *Oversampling.* The exchange of any data will automatically provide a level of synchronization among the channels. Processors will exchange data at given points and wait for all other processor's data. Thus data synchronization has an implicit real-time synchronization.†

† Just exchanging data does not automatically imply a lack of time divergence, a slow processor can still be "left behind". This is solved by various synchronization primitives to be discussed.

The combination of input consistency and some level of synchronization yields a very nice result. Aside from the data exchanges that must be employed on any input data and possible synchronization primitives, *the application program is written as if it will run on a single, highly reliable processor.* This means that the redundancy management techniques are truly application independent. The data synchronous system engineer provides a general purpose vehicle on which to execute a very broad class of problems, largely independent of the type of computations being performed. Conversely, the application's programmer is insulated from the vagaries of the implementation, and can write algorithms that will be *independent of the specific* vehicle on which it is to execute.

Redundancy management is distilled to the problem of maintaining input consistency and channel synchronization

5. CONSISTENCY MAINTENANCE

Maintaining input consistency is a demanding requirement. In theoretical circles, this problem is known as the *Byzantine Generals Problem,* and encompasses the discipline of reaching agreement in the presence of faults. Provable solutions to the problem exists, including several systems which have correctly implemented Byzantine agreement. In the following section we will develop a framework for the problem and its solution. In the sequel we shall translate this framework into tangible engineering designs and tradeoffs. It is essential that an engineer undertaking the construction of highly reliable systems understand the sources of inconsistency and be able to identify correct solutions. *The existence of consistency maintenance hardware is truly a litmus test for highly reliable systems.*

5.1. Threats, Faults, and Failure Independence

Any system is subject to threats to correct operation [9]. Threats take a number of forms: normal environmental threats such as component aging, abnormal environmental threats such as transients, or hardware/software design flaws. *Faults* are the anomalous conditions resulting from threats and a *fault model* is a functional description which captures the important aspects of expected faults.

Any design must assess the class of threats that the redundancy is expected to handle. The system is then partitioned into *fault sets,* each having independent failure statistics from the others. Fault sets are an abstraction of our intuitive notions of physical isolation. Generally speaking, *functions implemented on one processor* will be in a different fault set from functions implemented on another, provided suitable isolation is engineered between the processors.

The system redundancy management must prevent the propagation of faults across these fault set boundaries. Although a faulty processor can never directly cause another healthy to become faulty, it might very well cause another to compute irrational results. (Refer to the alpha vane problem presented in Section 3.5) The way a malfunctioning unit may influence good units is through the exchange of data and the exercise of control. Without loss of generality, we consider the exercise of control to be a kind of data exchange. Thus the effects of a malfunctioning unit upon a good one can be expressed purely in terms of information exchanges.

5.2. Modeling Data Transmission

It is precisely the vagaries of the information exchange mechanism that the fault model must capture and that our systems must deal with correctly. There are two central assumptions that permits a solution to the data consistency problem.

1. *The source of information is always known by a fault free receiver.* This has strong bias towards the use of interconnection links rather than multiplexed busses for the interprocessor data exchanges. The problem with not knowing the source of data (which fault set) *a priori,* is that a faulty processor might masquerade as different one. This does not prohibit multiple receivers on a link but imposes severe design problems when there are multiple transmitters. An important note: if multiplexed busses are used for the consistency exchanges then it is not guaranteed that any of the following results can be proven to hold. Beware!
2. *Information flows only from transmitter to receiver.* If full duplex communication is desired then a back link is required. Of course, this link might very well share the same physical medium. The restriction is really one from preventing a faulty receiver to influence a transmitter. This doesn't prohibit handshaking. It only requires that such mechanisms are *safe.* That is, if the receiver violates the signalling protocol, the transmitting processor is not somehow deadlocked or prevented from doing other computations.

The *modes* of failure that are of key interest to us are ones in which allow inconsistent data to be broadcast throughout the system. As such, we adopt the following aggressive and quite general fault model.

1. *A faulty processor may transmit any arbitrary stream of data any of its output links.* It may fail silently on some links, produce gibberish or babble on others, or even produce what looks like perfectly well formed messages but of arbitrary content.
2. *Two different receivers listening* to the same output may receive different *messages.* This is an essential property of any communication made through noisy channels with insufficient margins. The ability for a *failed transmitter* to "lie" is the most overlooked yet potentially catastrophic failure mode of all redundant systems.

It is easy to see how failures in the first case can readily occur in practice, and how such failure modes could lead to inconsistent system states. If processor A communicates with B over one link but with C over another it is perfectly possible that it could send different data B and C in the event of a failure in A.

The solution appears to be trivial. If it is so important that B and C get the same message from A then put them on the same link. The second aspect of the fault model would imply that this won't help. But how realistic is this assumption? How can two units reading the same wire get *different* results? Many different ways.

Figure 6 shows how this might happen. Suppose A communicates with B and C by broadcasting serial data and a decoding clock. If A fails in such a way that the signals on the data line become marginal then B and C could obtain different results. A's failure might be as simple as a bad solder joint on a terminator or a failing output transistor. In any case the event seems quite probable and it *must be considered in the design of any highly reliable system.*

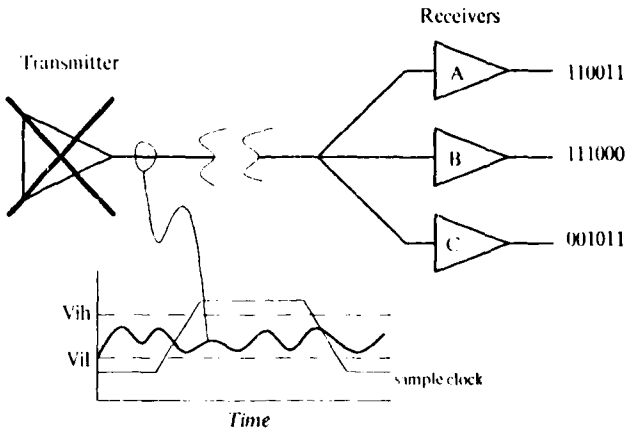


Figure 6. A Link Failure That Induces Inconsistent Messages

Even if heavy channel coding is used there is still the possibility that the receivers obtain different results. B might be able to recover the data while C might detect an unrecoverable error. The results are still different.

5.3. Exchange of Data Can Provide Consistency

In order to provide consistency it can be proved that data *must* be exchanged between processors. Figure 7 is a simple model for the distribution of a dual-redundant set of sensors to three processors. Fault sets are shown with dashed lines, so in this case the sensors are in their own fault set. Frequently, I/O devices are associated with a particular processor, so the input fault set might be the same as the processor (*i.e.*, if the processor fails then the sensor fails). We will deal with this case in a moment.

A simple cross strapping of these inputs, as shown in Figure 8 can lead to the types of inconsistencies described in the previous section. A failure of one the sensors such that it produces inconsistent data to the three processors could induce the divergence of all three processors.

Figure 9 shows how a single round of data exchange for each sensor can solve this problem. In the case of a failed sensor all of the processors exchange the value they obtained among themselves. The failure of the sensor is quickly identified because the final voting stage shows a complete inconsistency. Other failures of the sensor that do not lead to inconsistent outputs are caught by subsequent FDIR algorithms. Note however, that *all FDIR algorithms are using precisely the same data and will compute precisely the same results.* If a processor fails and broadcasts garbage information, this will be caught by that processor voting out. If this data exchange is performed on *every* unique input to the system, then all fault-free processors will have identical internal states. This, of course, assumes that the processors have the same initial state. A sticky issue that we will defer until the sequel.

Simultaneous Sensor and Processor Failure

Some systems get as far as providing these data exchange mechanisms, but with a serious defect. This defect arises when the processor and sensor fail simultaneously. Certainly a triplex system does not guarantee correct operation in the presence of two failures. Many times, however, the sensor is an I/O device of a particular processor. The processor samples the sensor, broadcasts the value to the other processors, and then enters a round of data exchange.

But if the processor has failed then it can produce inconsistent data during the initial broadcast and then again during the data exchange. Figure 10 shows how this can lead to the divergence of the two healthy processors. Certainly, processor hosted I/O devices need to be supported. Is there some other exchange mechanism that will work? Without another fault set the answer, unfortunately, is no.

Required Level of Redundancy

We are used to redundant systems working on the "majority principle"—a system containing $2f + 1$ processors can *sustain* f *simultaneous faults.* As long as there is majority of working processors we should be able, through simple voting logic, to separate the good from the bad. Results from testing theory bear out this lower bound.

Unfortunately, solutions to the Byzantine Generals Problem have proven that a system must contain at least $3f + 1$ failure-independent modules in order to support the consistent distribution of data in the presence of f simultaneous faults!

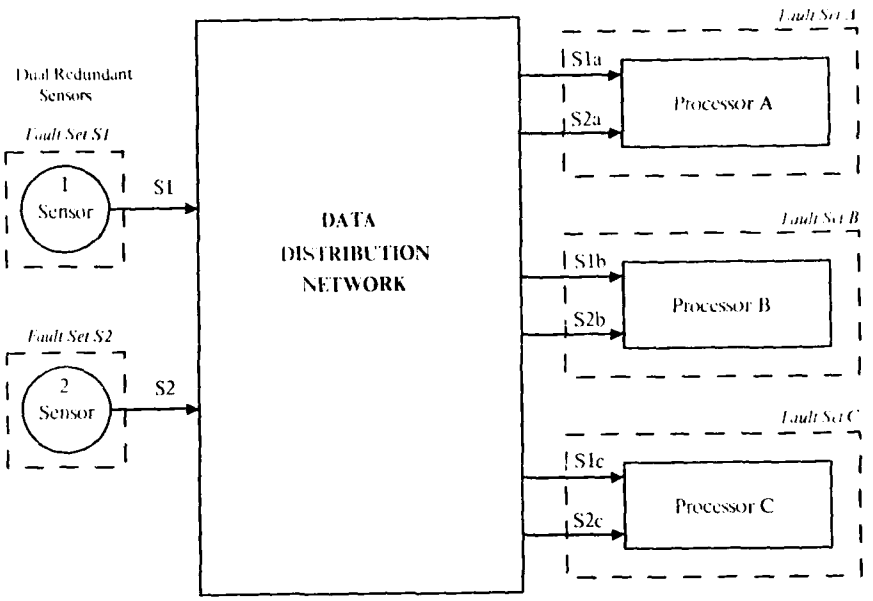


Figure 7. The Generic Problem of Distributing Redundant Sensor Data

Which bounds are correct? They both are, within their assumptions. It should be obvious that we really don't need more than $2f + 1$ independent processors to obtain a *majority* output, however, a total of $3f + 1$ are required to maintain consistent inputs. *The key observation is that all fault sets need not be processors.*

Restoring Stages

The root of the problem illustrated in Figure 10 is that the same fault set that sourced the original information also participated in the data exchange. This permitted the processor from corrupting the data a second time. Figure 11 shows a simple solution whereby the processor reads the information but sends it to separate fault set that subsequently performs the data exchange step. This simple device performs a *restoring* function on the original data. Its only function is to read its input and broadcast whatever it believed it read onto its outputs. In some sense this device is a simple repeater.

We don't mean to treat this property lightly. If self-clocking asynchronous data protocols are used the restoring function requires a great deal of care such that a bad input clock does not yield a bad output clock. Here, the restoring function must attempt to interpret the data and then rebroadcast it against its own clock reference.

Notice that this solution uses a total of six fault sets whereas the theoretical bounds only require four. The argument is that the restoring functions are quite a bit easier to implement than processors. This is the approach of Draper Laboratory's FTP which call the restoring functions *interstages*. Other designs, notably the Stanford Research Institutes SIFT system actually employs general purpose processors for the function.

Implied Synchronisation

These data exchange algorithms are predicated upon the assumption that all processors are somehow synchronised. This is the *stream* model of data presented in Section 2. If these input were allowed to be aliased among the different processors then the data exchange would clearly fail. More simply, think of the sensor as being sampled and converted into digital form. It is this *particular* sample that we wish to exchange and make consistent. Thus all processors must know precisely which piece of data is being exchanged. This can only happen if the processors are running frame or clock synchronously. In the case of frame synchronization, exchanges are expected to take place at given times during the frame cycle. In fact the data exchanges act as natural synchronisation points in the program. These are not sufficient, however. A correct method of processor synchronization will be give in the next section.

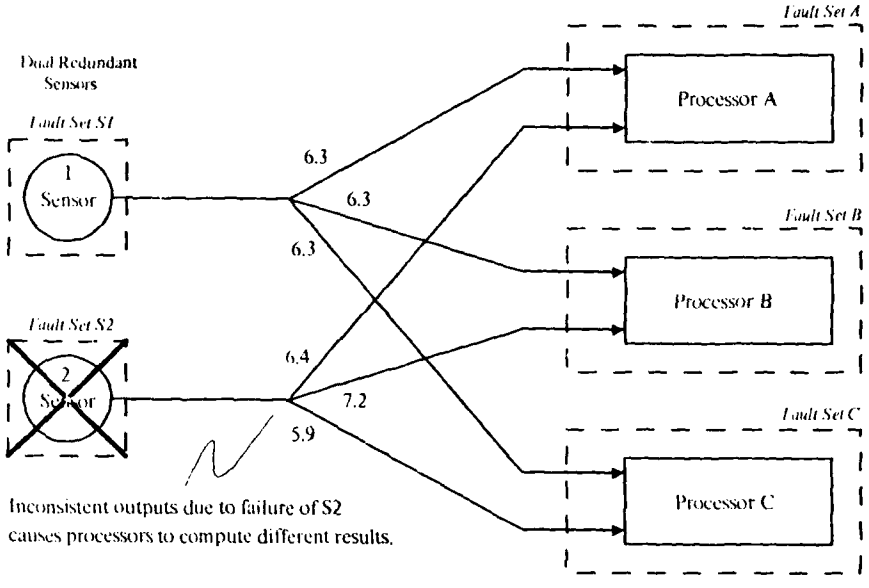


Figure 8. A Hazardous Solution to Sensor Distribution

6. *MAINTAINING* SYNCHRONIZATION

Suppose that in the course of a given computation the processors need to read a set of redundant sensors, say one sensor attached to each processor. For each sensor value a consistency exchange must take place. Because the processor must use results from one another, they must wait until all of the answers from the current exchange have been received. Some processors may reach the *exchange point* sooner than others. How long should a processor wait for that is not responding? If some hard *timeout* limit is imposed then we must ensure that only faulty processors will timeout. This is the *synchronisation problem.*

6.1. A Common Design Flaw

Common solutions involve the exchange of a synchronisation discrete. The F-8 DFBW system employs the following *synchronization* algorithm [10].

> "Every 20 milliseconds an internal clock interrupt occurs and the computer issues a discrete high signal to each of the other computers. The computer then reads the discretes it has received from the other computers, the discrete is reset, and a second read is performed to ensure that the other computers have also reset their discretes. This process accomplishes synchronisation. The computer clock is reset to interrupt the next cycle time. If, after a short wait to allow for skew between the processors, one computer fails to synchronise with the other two, the two remaining computers exit the synchronisation program and continue normal processing."

Although it is noted that the algorithm performed well under quite extensive tests, it does possess a single-point failure mode. Figure 12 shows a simple example where a failure in one processor can cause the other two to lose synchronisation.

In this example, processor *A* is normally the slowest, followed by *C* and then *B*. Normally, the phase-locking property of the above algorithm keeps all processors in synchronisation, paced by the slowest processor, *A*. Suppose that *A* experiences difficulty with its Input/Output subsystem—perhaps a bad power supply. This causes *A* to put out marginal signals in such a way that *C* still sees *A's* synchronisation discrete, but *B* only sees spurious signals whenever it enters the synchronisation loop.

From *B's* perspective processor *C* is slowest so *B's* timing is derived from *C* as long as *C* is within the timeout window. From *C's* perspective processor *A* is the slowest so it's timing is derived from *A*. It is now perfectly possible for the slowness of *A* to pull *C* out of *B's* timeout window. When this happens *B* considers *C* failed and just synchronises

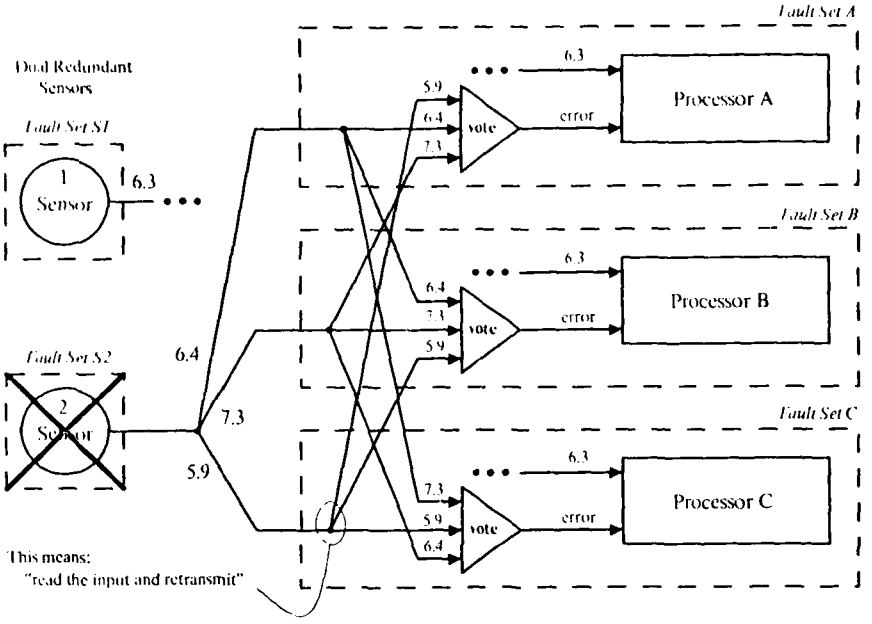


Figure 9. Data Exchange Provides Correct Sensor Distribution

with *A*. Similarly, *C* considers *B* failed and also synchronizes with *A*. Now suppose that *A* finally fails hard over—the entire system loses synchronization.

This isn't a problem with the algorithm. *No* algorithm can be free from single point synchronization failures with only three fault sets (processors). The problem is inherent with the system architecture.

6.2. A Correct Implementation of Multiprocessor Synchronization

The problem of distributing time in a system subject to faults is related to the Byzantine Generals problem and is also solvable. The solution is remarkably similar.

Suppose we slightly modify the synchronizer described above such that it selects mid-value edge to synchronize from. We then introduce a delay equal to the maximum expected skew before proceeding. This is show in Figure 13. In the case where all synchronization discretes are *consistently* distributed, the processors resynchronize to levels equal to the intrinsic skew in the synchronizer, not the original skew among the processors. Notice that a faulty processor, however, can induce the two healthy processors to miss-synchronize only by an amount equal to the skew between the good processors. The problem is that after each synchronization this error is accumulated, ultimately driving the good processors out of synchronization.

As in the case of data exchanges, we can correct for this inconsistency by introducing a second level of synchronization. This is shown in Figure 14. The analysis of this is quite simple. A single failure can only affect one or the other synchronization exchanges. If the first level has a problem then the discretes presented to the second level will be bounded by the skew among the healthy processors.

The two-level algorithm fits nicely with the restoring stage concept. The first set of mid-value selectors is implemented by the restoring stage. The second set occurs back at the processors. From an engineering standpoint the topology of the interconnection is no more complex then that required to support data exchanges from any processor, although a *particular* exchange only uses a subset of the first set of communication links. It is even possible to make all data exchanges so that they automatically resynchronize the processors. We defer the discussion of such tricks to the sequel.

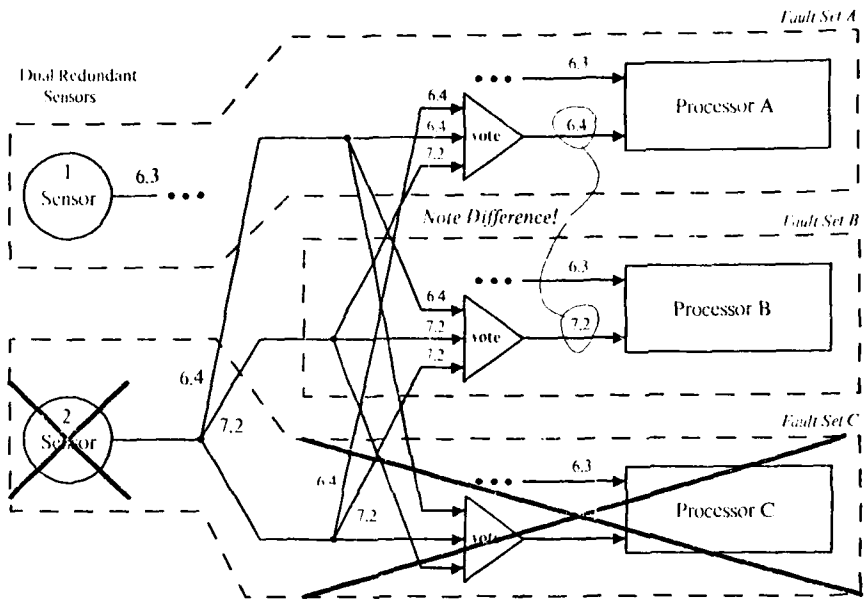


Figure 10. Simultaneous Sensor and Processor Failure is Disastrous

7. CONCLUSION

The management of sophisticated modular redundant systems is really a problem of managing complexity. It is possible to handle the complexity by suitably decoupling the problems of developing algorithms for a highly reliable system from those of providing a correctly operating vehicle on which to execute these algorithms.

Systems that do not require exact agreement among redundant channels, what we have termed data asynchronous, do not permit this decomposition. Data synchronous systems, however, can provide general purpose techniques for managing digital computer redundancy that lead to policies that are independent of the target applications. Certainly a control system is not just composed of computers and their interconnection. Redundancy management of sensors and actuators must always be performed—but these solutions can be made assuming that the algorithms themselves will always execute correctly.

Systems whose redundancy management is sensitive to the type of applications be performed will have great difficulty adapting to more and more complex functions. Data synchronous systems provide the only viable path for future systems.

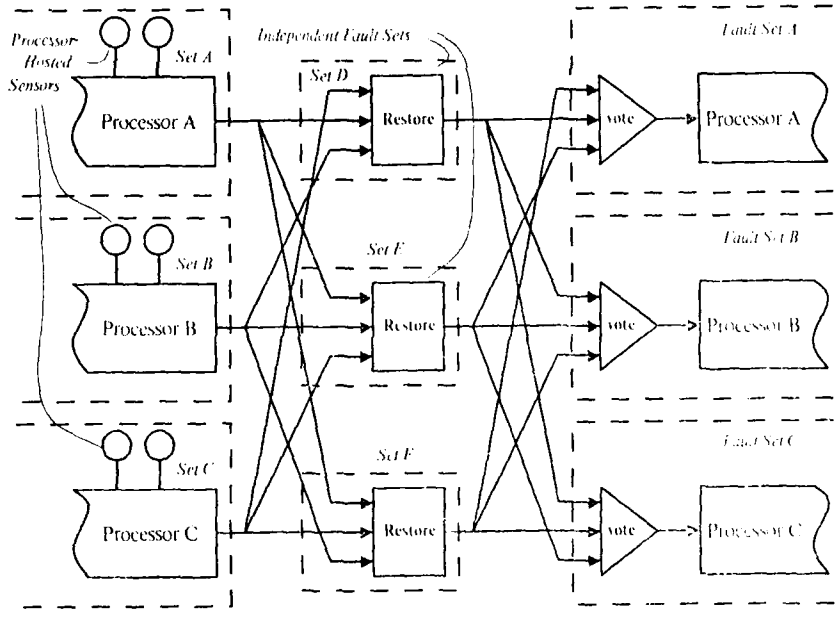


Figure 11. Dedicated Restoring Stages Add Hardware But Improve Performance.

REFERENCES

[1] G. M. Papadopoulos, "Design Issues in Data Synchronous Systems," *Agard Lecture Series No. 143*, October 1985

[2] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 27, No. 2, April 1980, pp. 228–234.

[3] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982, pp. 382–401

[4] A. L. Hopkins, Jr., *et al.*, "FTMP—A Highly Reliable Fault-Tolerant Computer for Aircraft," *Proceedings of the IEEE*, Vol. 66, No. 10, October 1978, pp. 1221–1239

[5] D. Davies, J. F. Wakerly, "Synchronization and Matching in Redundant Systems," *IEEE Transactions on Computers*, Vol. C-27, No. 6, June 1978, pp. 531–539

[6] L. Lamport, "Time, Clocks, and Ordering of Events in a Distributed System," *Communications of the ACM*, Vol. 21, No. 7, July 1978, pp. 558–565

[7] W. M. Daly, A. L. Hopkins, and J. F. McKenna, "A Fault-Tolerant Digital Clocking System," *Dig. 3rd Int. Symp. Fault-Tolerant Computing*, IEEE Publication 73CH0772-4C, June 1973, pp. 17–22.

[8] T. B. Smith, III, "Synchronous Fault-Tolerant Flight Control Systems," *C. S. Draper Laboratory Report P-1404*, 1982

[9] A. L. Hopkins,"Fault-Tolerant Systems Design: Broad Brush and Fine Print," *IEEE Computer*, March 1980, pp. 30–45.

[10] K. J. Szalai, R. R. Larson, R. D. Glover, "Flight Experience with Flight Control Redundancy Management," *Fault Tolerance Design and Redundancy Management Techniques*, AGARD Lecture Series No. 109, October 1980

This paper was set in Computer Modern Roman 10 point by the author using the TEX typesetting system. The paper was printed on an Epson FX100+ driven by an IBM PC/XT. Line drawings were created with ILLUSTRATE on a Symbolics 3670 Lisp Machine and printed on a Xerox Dover laser printer.

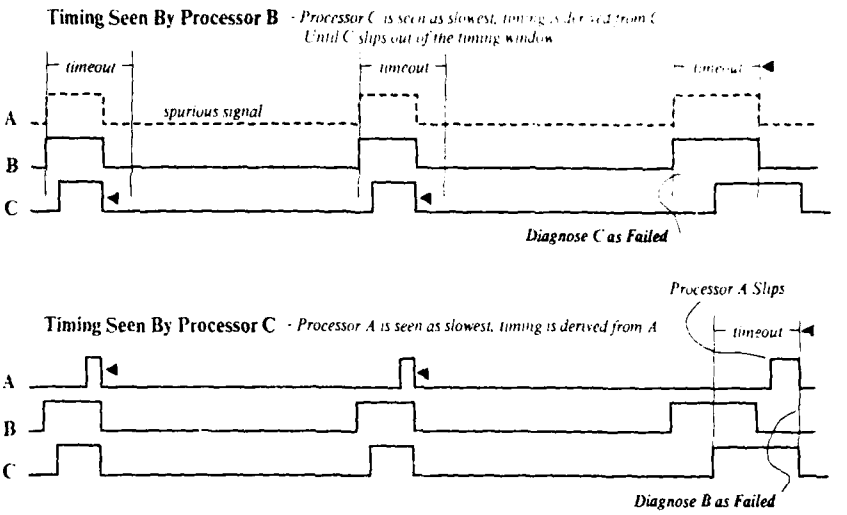


Figure 12. Single Point Synchronization Failure in the F-8 DFBW

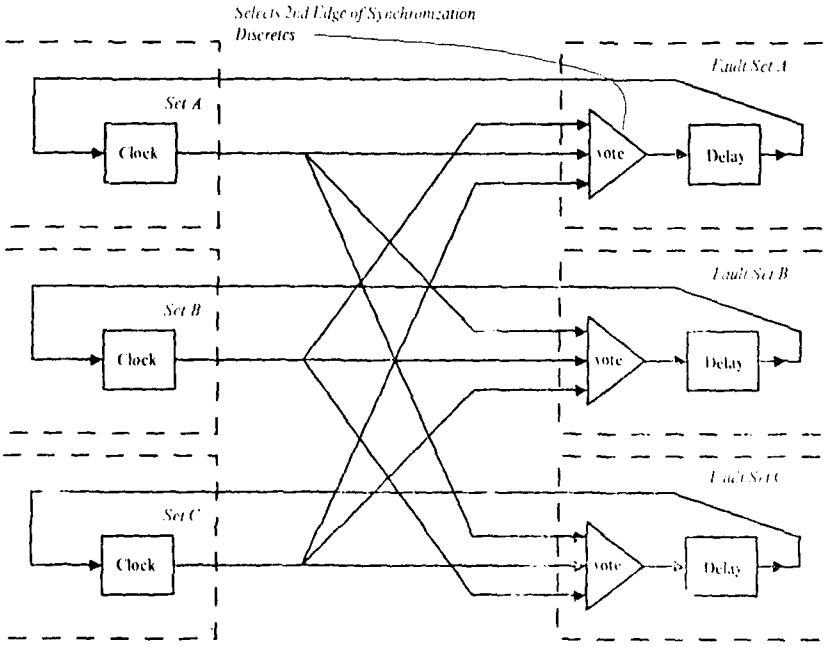


Figure 13. A One-Level Synchronization Algorithm

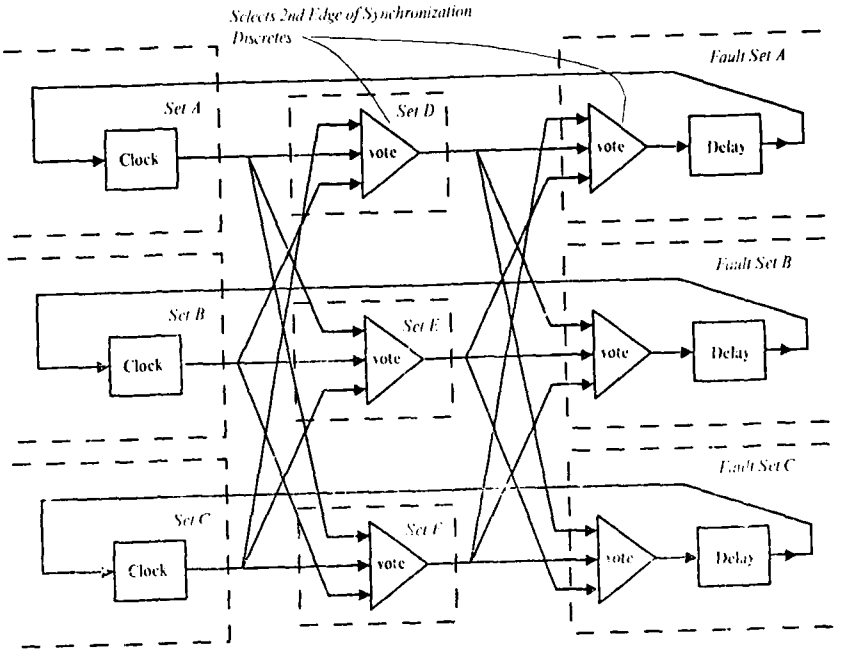


Figure 14. A Two-Level Synchronization Algorithm

Software Fault Tolerance Experiments

T. Anderson and P. A. Barrett

Centre for Software Reliability
The University of Newcastle upon Tyne
Computing Laboratory
Claremont Tower
Claremont Road
Newcastle upon Tyne NE1 7RU
England.

June 1985

Summary

A major experimental programme has been undertaken at the University of Newcastle upon Tyne in order to evaluate the effectiveness of software fault tolerance techniques in practical systems. This paper presents the results of phases two and three of these experiments, which indicate that the techniques can significantly enhance software reliability. The particular application used for these experiments was a naval command and control system, thus confirming that software fault tolerance can be successfully utilised in critical real-time systems.

Introduction

The process of software development is usually described in terms of a progression from user requirements to the final code, passing through intermediate stages such as specification, design and validation. Of course, progress through these stages is rarely unidirectional, and "final code" must be considered to be a misnomer given the demand for subsequent software maintenance. An engineering approach to software develoment should enable software to be produced on time, within budget, and in accordance with user requirements. One important aspect of these requirements concerns the **reliability** of the software. Software reliability requirements can be expressed in a number of ways, of which the simplest is perhaps to impose an upper limit on the measured rate of failure over a specified interval.

Given that reliability criteria can (and should) be imposed on software systems, how can these standards of reliability be achieved? Fortunately there is a wide range of techniques available to the software developer, all intended to enhance software reliability. These techniques may be categorised as follows [7]:

1. Techniques to avoid making mistakes - such as design methodologies and notations - referred to as **fault avoidance.**

2. Techniques to find and remove mistakes - such as design reviews, code inspection, program analysis, testing, verification, all followed by debugging or redesign - referred to as **fault removal.**

3. Techniques to cope with mistakes - defensive programming based on redundancy - referred to as **fault tolerance.**

The major obstacle impeding the construction of reliable software according to engineering principles is the shortage of data on the effectiveness of these various techniques. This is particularly the case for software fault tolerance techniques, where experience is limited and most experimentation has been confined to relatively small modules. However, a recent paper [1] reported the results of a first phase of experimentation with a system of realistic size implemented using software fault tolerance. This paper presents the results from two further phases of experimentation from the same project.

Overview of Project

Only a brief summary is presented here; further details may be obtained from project reports [2,3]. In order to evaluate the effectiveness of software fault tolerance techniques a realistically scaled software system was implemented by professional programmers to approved commercial and military standards. The actual application selected provided a subset of the functions of a naval command and control system. Three types of sensor input were used to generate tracking information on objects in a simulated tactical environment. This information was presented to a human (or partially automated) operator via visual display units. The operator could then initiate a "VECTAC" - a vectored attack on a hostile submarine by means of a torpedo launched from a helicopter. The command and control system comprised about 8000 lines of CORAL code running under the control of a MASCOT operating system [4]. This executed on a PDP-11/45 and interacted with the environment simulator system running on an LSI-11/23. A second PDP-11/45 provided data recording facilities and other support services. MASCOT provides pseudo-parallel execution of concurrent activities (14 separate processes in this system). Activities can only communicate by means of shared data areas maintained by

MASCOT according to the system designer's stipulations on access paths and methods.

Software fault tolerance was incorporated in the command and control application software, in the form of acceptance tests and alternate modules. These were used to provide recoverable "dialogues" between activities. A dialogue is an explicit embodiment of, and notation for, a restricted form of the concept of a "conversation" [5] which is, in turn, an extension to concurrent systems of the recovery block technique for software fault tolerance [6]. Further information on these, and other, methods of providing tolerance to software faults is available elsewhere [7,8].

The MASCOT operating system was modified and extended to provide recovery capabilities for activities and for information recorded in the shared data areas. These recovery mechanisms utilised a special purpose hardware device, called the recovery cache [9], which enables state restoration to be performed very quickly (the recovery cache may be thought of as providing a highly optimised implementation of checkpointing for multiple processes).

Experimental Programme

In order to measure the effectiveness of the software fault tolerance techniques in enhancing reliability a series of experimental runs were performed using various tactical scenarios to drive the simulator system. Three phases of experimentation were conducted. The results and analysis of the first phase have been reported [1,2]; this paper presents the data obtained from the second and third phases and analyses this data to obtain an assessment of the effectiveness of software fault tolerance.

For each phase of experimentation the application software was frozen; that is, no changes were made to the command and control software during a phase of the experiments. However, the first phase of experiments involved two versions of the command and control system. In version one the software fault tolerance was enabled and operated normally, whereas in version two, fault tolerance was disabled by the simple expedient of forcing all run-time checks to return a positive (ie ok) response. Thus a comparison between the two versions enabled a direct confirmation to be obtained of other measurements made of the improvement in software reliability. For phases two and three it was felt that our knowledge of the system was adequate to dispense with this confirmation, so all runs were performed with fault tolerance enabled.

In the second phase of experiments the same command and control software was used as for the first. In part, the intention was to confirm the results of phase one. More importantly, however, the first phase identified numerous problems with the MASCOT recovery software, and these were corrected for phase two. Since the success of the fault tolerance techniques is dependent on the recovery mechanisms, the results from phase two should more accurately reflect the benefits possible from fault tolerance in practice.

In the third phase of experiments, the command and control software was modified by replacing a number of modules with new versions written by inexperienced programmers. These versions were expected to contain a greater number and wider range of faults than the original modules. Furthermore, where original modules were retained, the sequencing of alternates in recovery blocks was reversed, so that the back-up alternates were used as primary alternates (and vice-versa). Any faults in the recovery mechanisms identified during phase two were rectified before phase three.

Two further phases of experimentation were envisaged, and one of these was attempted. The intention was to evaluate the effectiveness of the fault tolerance techniques at higher levels of software reliability, and to this end, all faults identified in the application system during phase one were rectified to yield a more reliable version of the command and control software. Unfortunately, this system proved too reliable, in that failure data was generated much too slowly. This phase of experimentation was therefore terminated unsuccessfully.

Time and financial limitations precluded the last phase of experimentation, in which it was planned to utilise an unreliable version of the application system derived from incompletely tested modules which had been archived during the development of the command and control software.

Each phase of experiments consisted of a number of runs (60) of the command and control system in which tactical scenarios were enacted on the simulator. Each run was monitored by the support system, and was carefully observed by an operator. Each time an event occurred (an event is either a system failure, or the detection of a real, or imagined, error in the state of the system) the entire system would halt, and the operator would analyse the error and attempt to identify the fault which caused it. The run would then be continued to see if fault tolerance would enable a failure to be averted, or if the failure would nevertheless occur. A run was considered to have finished when the scenario was completed, or when a failure occurred which prevented the system from continuing.

Experimental Programme Results

In order to analyse the data from each run it was necessary to determine whether or not each event would have resulted in failure had the system contained no fault tolerance features. Usually the answers to such questions were obvious, but had there been any doubt surrounding the outcome of a particular event in a non-fault tolerant system, the

option was available to run the system in non-fault tolerant mode and attempt to re-create the event in question. The effects of the event would then be directly observable. This was not found to be necessary in phases two and three of the experiments.

The following categories were used to group events:

1. Events which produced recovery which averted failure.
2. Events in which recovery occurred unnecessarily, but no failure resulted (usually a consequence of a faulty acceptance test).
3. Events in which a successful recovery took place, but the system failed nevertheless, as it would have done in the absence of fault tolerance.
4. Events in which recovery was attempted but was not successfully accomplished, and the system failed, as it would have done in the absence of fault tolerance.
5. Events in which defective recovery **caused** the system to fail.
6. Events in which the effect on the system is unclear.

Events in category 1 yield an improvement in reliability due to fault tolerance whereas those in category 5 result in a deterioration in reliability (those in categories 2, 3 and 4 do not affect reliability).

Two cases are considered; firstly a summary of all events, and secondly a summary of the first events which occurred in each run. This distinction is made to factor out any effects which might arise due to including events which occur **after** a non-fault tolerant system would have failed.

Summary of All Events	**Phase 2**	**Phase 3**
1. Recovery averting failure	34	91
2. Unnecessary recovery	6	4
3. Recovery followed by failure	5	4
4. Defective recovery	18	17
5. Failure caused by recovery	4	0
6. Outcome unclear	1	1
Total Events:	68	117

Summary of First Events	**Phase 2**	**Phase 3**
1. Recovery averting failure	20	24
2. Unnecessary recovery	3	4
3. Recovery followed by failure	1	0
4. Defective recovery	10	5
5. Failure caused by recovery	3	0
6. Outcome unclear	0	1
Total first events:	37	34

Analysis of Results

The principal measure of the effectiveness of software fault tolerance was taken to be the "coverage" factor of these techniques; that is, the proportion of failures which would have occurred in a non-fault tolerant system which would be successfully averted by means of fault tolerance. To be more precise, for situations in which the non-fault tolerant system would fail, **coverage** represents the probability that the fault tolerant system will nevertheless continue to operate without failing. The required probability can be easily estimated from the data of the previous section and thus relies solely on event counts. The coverage factor is calculated as the ratio of the number of failures averted (event category 1) to the number of potential failures (event categories 1, 3 and 4). Events in category 2 (spurious recovery) and category 6 (unclear events) are disregarded. Events in category 5 (failures introduced by fault tolerance) cannot be ignored, but are excluded from the initial calculation.

Thus, considering all events in phase two of the experiments the coverage achieved by fault tolerance is estimated to be 34/57, which is approximately 0.60. This is the

maximum likelihood estimate. A Bayesian analysis using the Beta distribution indicates that the value estimated can be asserted to exceed 0.52 with 90% confidence. These figures should be abated to take into account the four failures caused by fault tolerance. The simplest approach regards these failures as "own goals" and subtracts them from the successes of category 1. An amended coverage estimate of 0.53 is then obtained.

The following table presents these coverage estimates for all three phases of experimentation, those for phase 1 being included for comparison and completeness. The estimates have been calculated for the two sets of data, namely all event data and first event data.

Failure Coverage by Software Fault Tolerance

	Phase 1	Phase 2	Phase 3
All Events			
Raw Coverage	0.75	0.60	0.81
Bayesian 90% point	0.67	0.52	0.77
Abated Coverage	0.68	0.53	0.81
First Events			
Raw Coverage	0.44	0.65	0.83
Bayesian 90% point	0.29	0.53	0.74
Abated Coverage	0.25	0.55	0.83

To obtain an absolute measure of system reliability, the execution time of the system was recorded for each run and summed over each phase of experimentation. This enabled the failure rate for the command and control software to be estimated as 1.36 per hour in phase 1, 0.88 per hour in phase 2 and 0.58 per hour in phase 3. A coarse comparison can be made with the failure rate of the non-fault tolerant system in phase 1, which was 3.21 per hour.

Discussion and Conclusion

The results of the previous section show clearly that for this application, in these experiments, the incorporation of software fault tolerance has yielded a substantial increase in reliability. Over the entire programme of experiments, the event counts show that 222 failures could have occurred due to "bugs" in the software of the command and control system. But of these 222 potential failures only 57 actually happened - the other 165 were masked by the use of software fault tolerance. This represents an overall success rate of 74%. (The same calculation restricted to first events yields the slightly lower figure of 67%.)

Examination of the results from the first phase of experiments [1] suggested that much better results could be achieved if the underlying recovery mechanisms could be brought to an adequate standard of reliability. Essentially, the project was relying on prototype recovery mechanisms (the recovery cache and the MASCOT recovery software) to support the provision of fault tolerance at the application level. This situation would most certainly not be typical of an operational system where the recovery facilities should be at least as reliable as the hardware itself. It was hoped that improvement to the recovery routines for phase 2 would produce improved results, but in fact this effect was not observed until phase 3. Projections suggest that with further improvements to the recovery software a coverage factor of over 90% could have been achieved.

The discrepancy between the results for all events and first events is very marked for phase 1 but is minimal in phases 2 and 3. The most likely explanation is that the all events results for phase 1 are rather better than they would otherwise be as a result of multiple recovery successes occurring in sequence. This phenomenon did occur in one spectacular case in phase 1 where a series of 12 successful recoveries in rapid succession helped boost the figures (and, to some extent, project morale).

Of course the reliability gains were achieved at a cost, paid in capital costs to support fault tolerance, development costs to incorporate fault tolerance and run time and storage costs to make use of fault tolerance. These costs are presented and discussed in the earlier papers [1,2]. Very briefly they involved 1000 hours of capital development, 60% supplementary development cost for the command and control system, 40% run time overhead and 35% extra storage. All these figures (except perhaps the last) are likely to be on the high side, reflecting the novelty of the techniques, their widespread use for this experiment, and the omission of any fine tuning or optimisation of the system.

Our overall conclusion is that these experiments have shown that by means of software fault tolerance a significant and worthwhile improvement in reliability can be achieved at acceptable cost. We look forward to an independent confirmation of this result, preferably in the context of a system to be used in earnest.

References

[1] T. Anderson et al., **An Evaluation of Software Fault Tolerance in a Practical System**, To appear in Digest of the 15th. Fault Tolerant Computing Symposium, Ann Arbor, 1985.

[2] T. Anderson et al., **Fault Tolerance Project Report: Results and Conclusions from the Experimental Programme**, Project Report ref. 4844/DD.17/2, University of Newcastle upon Tyne, 1984.

[3] T. Anderson and P. A. Barrett, **Fault Tolerance Project Report: Results and Conclusions from the Second and Third Experimental Programmes**, Project Report ref. 4844/DD.17/3, University of Newcastle upon Tyne, 1985.

[4] Mascot Suppliers Association, **The Official Handbook of MASCOT**, Royal Signals and Radar Establishment, Malvern, 1980.

[5] B. Randell, **System Structure for Software Fault Tolerance**, IEEE Transactions on Software Engineering, SE-1(2), pp.220-232, 1975.

[6] J. J. Horning et al., **A Program Structure for Error Detection and Recovery**, pp. 171-187 in Lecture Notes in Computer Science 16, Springer-Verlag, 1974.

[7] T. Anderson and P. A. Lee, **Fault Tolerance: Principles and Practice**, Prentice Hall, 1981.

[8] T. Anderson, **Can Design Faults be Tolerated?**, pp.426-433 in Fehlertolerierende Rechensysteme, Informatik-Fachberichte 84, Springer-Verlag, 1984.

[9] P. A. Lee et al., **A Recovery Cache for the PDP-11**, IEEE Transactions on Computers, C-29(6), pp. 546-549, 1980.

Acknowledgements

This work was supported by the UK Science and Engineering Research Council and the Procurement Executive, Ministry of Defence. Without the assistance of our colleagues D.Halliwell and M.Moulding these experiments could not have been conducted.

Can Design Faults be Tolerated?

T. Anderson

Centre for Software Reliability,
Computing Laboratory,
University of Newcastle upon Tyne.
December, 1984

Abstract

The fault tolerant approach to building a reliable system acknowledges that perfection is impossible (or at best, very expensive) and therefore tries to cope with the consequences of residual defects within the system. Fault tolerance has an established role in detecting and masking component faults in hardware systems, but has also been advocated as a defence against deficiencies of design. This paper argues, in question and answer format, the case for adopting design fault tolerance techniques in practical systems.

Introduction

The short answer to the question posed by the title is "Yes". A more cautious, and less simplistic, response would be that in certain circumstances, with appropriate provision of redundancy and allied supporting mechanisms, it is certainly possible to provide a measure of tolerance to faults of design. However, although this question may serve as an appropriate title, and starting point for discussion, it does not adequately address the significant issues concerning the application of fault tolerance techniques to deficiencies of design. As is usually the case, the first, and perhaps most important, step is to ask the right questions. In this paper, I propose to substitute five further questions in place of my title and, in answering those questions, will argue the case for the use of design fault tolerance in the development of reliable computing systems. In so doing I hope to justify the short and cautious answer already given above.

The following discussion will largely be conducted with reference to the use of design fault tolerance in software systems, since current techniques were devised primarily for use in software development. It is however, customary (and accurate) to make the observation that the increasing complexity of VLSI designs suggests that software design techniques may also have a valuable contribution to make in the area of hardware design.

Is there any need for design fault tolerance?

The traditional approach to achieving reliability in computing systems has largely been based on fault prevention, the goal of which is to prevent system failure by ensuring that no faults can be present when the system is in operation. There are two aspects of fault prevention, which may be termed fault avoidance and fault removal.

Fault avoidance concerns those techniques which aim to avoid the introduction of faults during the design and construction of a system. Since this approach may not be completely successful, fault removal techniques are necessary to validate the implementation of a system and remove any faults which are thereby exposed.

If fault prevention is not expected to completely eradicate faults from a system, then fault tolerance techniques can be employed to provide a last line of defence. By incorporating redundant elements it may be possible to cope with the effects of a fault during system operation, and thus avert the occurrence of a failure (1).

The provision of tolerance to anticipated hardware faults has been a common practice for many years, for two reasons. First, hardware is by its nature built from physical components, and these are susceptible to the introduction of faults arising from the natural processes of decay and deterioration in the physical realm. Second, the effects of physical faults can often be categorised into well understood "failure modes" for components, which greatly assists the selection of appropriate redundancy. The first reason establishes the need for fault tolerance in hardware, while the second facilitates its provision.

For software the situation is rather different. Software is by its nature abstract rather than physical and not subject to faults introduced during operation by "*software rot*" (contrary to popular belief). Of course, the representation of the software may be corrupted by the effects of a hardware fault, but that is a separate issue. Thus, any faults in the software itself are design faults, due to mistakes made during the development process which escaped the vigilance of fault prevention techniques. It follows that the effects of software faults are difficult, if not impossible, to anticipate - which makes the implementation of techniques to contend with those effects somewhat more demanding.

Nevertheless, there have been a number of proposals which advocate the use of fault tolerance in software. I would argue that the need for such techniques is, in principle, self evident. A wide range of techniques are available for fault prevention in software (including notations for requirements, specification and programming, design methodologies, validation and verification, management and support environments ...) but though these may be highly beneficial, they certainly do not eliminate all faults from programs. Future software engineering developments may enable us to achieve such high standards of software design that fault tolerance has no role to play, but I suspect this will only be the case when either mechanically checked formal verification is possible and economical for practical systems, or software can be generated automatically from specifications. Even then, the problem of inadequate or inaccurate specifications will remain.

Current techniques for building reliable software systems rely largely on "exhaustive testing", that is, testing continued until either the project budget, or the software tester, is completely exhausted. The diminishing return on investment from such testing argues forcefully for the adoption of a wider range of techniques - which could sensibly include design fault tolerance.

Is design fault tolerance a mature technology?

A recent study in this area concluded that "all evidence indicates that fault-tolerant software technology has progressed sufficiently ... to move out of the laboratory into practical systems" (2, section 1.4). That is, the technology of design fault tolerance has been extensively developed in theoretical and experimental contexts, and is now ready to be adopted in practical systems.

To provide tolerance to design faults, redundancy must be extended to cover the design. Avizienis terms this redundancy "design diversity" (3). Two different approaches have been proposed, namely recovery blocks (4) and multi-version software (5). Although these are often viewed as distinct (even competitive) methods, the differences are principally implementation issues rather than major conceptual matters. Indeed, an elementary generalisation can be presented which encompasses both approaches.

Suppose we have a software module M_1 which receives input and produces output as shown in figure 1.

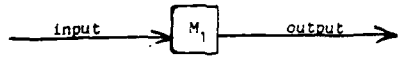


Figure 1. Single Module

To provide tolerance to possible design faults in M_1 we supply independently designed alternative versions $M_2,\ldots,M_n$ and apply one or more of these to the input. We must decide which of the n possible outputs is actually to be used and so a selection must be made by an adjudication module A. This is depicted in figure 2.

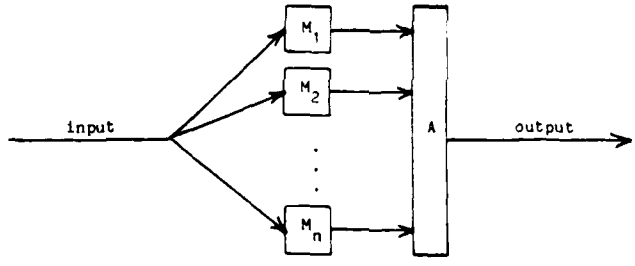


Figure 2. Design Diversity

The simple structure of figure 2, highly reminiscent of hardware NMR structures, is sufficient to represent either recovery blocks or N-version programming. The only substantive generalisation is that the form of the adjudication algorithm has not been specified.

In N-version programming, module A uses either simple majority voting, or inexact voting when permitted tolerances on outputs preclude a unique correct output. Recovery blocks usually apply a fixed acceptance test to the output in a predetermined priority sequence (Lee (6) suggested variant forms of acceptance test). Other adjudication algorithms are possible, of course, and have been suggested by the authors of hybrid schemes (7,8).

The standard descriptions of recovery blocks assume sequential execution of $M_1,\ldots,M_n$ when required, with the ability to regenerate an initial state, whereas N-version programming was envisaged as employing parallel execution on multiple processors, with state replication. From a strict semantic viewpoint, these are mere details of implementation. N-versions could be executed serially, just as the recovery block alternates could be performed in parallel.

The study of fault tolerant software (2) quoted above summarised the results of seven attempts to develop software reliability models (most recently by Scott et al. (7)) for design fault tolerance notations. Much of this work suffers from a lack of empirical validation, and depends heavily on assumptions which may be questioned. Nevertheless, all the models do confirm the potential for reliability enhancement which design fault tolerance offers.

Other recent work has addressed the applicability and effectiveness of design fault tolerance in real-time systems (9, 10) and in concurrent systems (11). Cristian has continued work on the interrelationship of fault tolerance and exception handling mechanisms and notations (12). A lot of earlier work on design fault tolerance has been summarised elsewhere (1). It can surely be claimed that the conceptual development of design fault tolerance has received considerable attention and is now well understood. But is it of relevance to the implementation of systems in practice?

Can design fault tolerance be utilised in practical systems?

There are many instances of the use of design fault tolerance in the software of practical systems with high reliability requirements. However, this use is often ad-hoc, unstructured and of limited fault tolerance capability. It is usually referred to as defensive programming and is often flagged in the software with the comment "This should never be executed but ...".

Multi-version software has been developed for a small number of practical systems, usually at the insistence of the relevant regulatory authority that the software should not constitute a single point of failure. Examples are the slot and flap control system of the Airbus A310 (13) and the flight control system of the Boeing 737-300 (14). Both of these systems employ dual dissimilar versions of the entire software. Outputs are compared, and if a discrepancy is detected the systems revert to a passive mode of operation, alerting the flight crew. Similar approaches have been adopted in systems for railway signalling (15) and nuclear reactor shutdown.

Perhaps the best known instance of design fault tolerance is that used in the NASA Space Transportation System, the "Space Shuttle" (16). A single backup computer runs in parallel with four primary computers. The primary computers execute the normal software system whereas the backup computer executes an alternative version of software for mission critical functions. Error detection is performed by comparison, voting, built in self-checking, and the ultimate acceptance test - the astronauts themselves. A switch-over to using the back up software can only be initiated manually by the crew.

None of these practical systems makes use of design fault tolerance in the modular and hierarchical fashion which is possible using recovery blocks or N-version programming. Hierarchical use of recovery blocks has been achieved in a research project at Newcastle (15), which has implemented a software system of realistic scale to assess the effectiveness of design fault tolerance techniques (17). The actual application selected supported a subset of the facilities of a naval command and control system, and was implemented in accordance with commercial practice by experienced programmers. Approximately 8,000 lines of program (written in CORAL) generated nearly 50 Kbytes of machine code.

Since the application was designed as a concurrent real-time system containing 14 separate processes, it was necessary to devise a notation and mechanism supporting a form of "conversation" (4,11) which would coordinate the recovery capability of interacting processes. The resulting structure was called a dialogue (since this means a conversation of a formal or restricted nature) and will be described in a forthcoming paper (18). Essentially, dialogues are used to define multiprocess recovery blocks which are statically nested at compile time. Additional features facilitate their use in cyclic computations.

An earlier form of dialogue, based on dynamic process structuring, met with little success and was quickly replaced by the static form. Thereafter, the only difficulty encountered by the system developers was in devising the acceptance tests needed to provide run-time error detection. Many of these tests were selected without difficulty, but certain situations caused problems. These were resolved by resorting to a structural consistency check of primary data structures. The overall conclusion of the system developers was that the design fault tolerance techniques, though novel, were certainly usable in building a practical system.

Can design fault tolerance improve system reliability?

Very little information is available as yet on the effectiveness of multi-version software in practical systems, though most projects report that construction of dual versions was of great assistance during development as a means of simplifying testing procedures. Similarly, only limited encouragement can be drawn from the reliability models for design fault tolerance mentioned earlier. Of course, the Space Shuttle diverse software provided perhaps the most famous bug ever recorded (16) by failing to synchronise, and aborting the first launch. (Note that the fault tolerance operated flawlessly on that widely publicised occasion.)

Experimentation at UCLA with N-version programs (19) involved the implementation of 18 versions of an airport scheduling program by students. All triad combinations of these versions were evaluated as 3-version programs. In 27.1% of these combinations, two correct versions succeeded in masking the faulty computations of a defective third version. Only in 2.5% of the cases was an incorrect result produced. These experiments confirmed the positive results of preliminary evaluation studies (5) on N-version programming.

Our aim at Newcastle in applying fault tolerance to the design of a naval command and control system was to obtain a quantified evaluation of the effectiveness of design fault tolerance techniques in the context of a practical system. When the software had been thoroughly tested and was considered ready for "operational" use, a lengthy series of experimental runs was performed using a simulated tactical environment and a variety of action scenarios.

A detailed analysis of these runs showed that on 53 occasions the software would have failed in the absence of fault tolerance, but by means of fault tolerance, failure was averted in 40 of these situations. Thus a failure coverage of 0.75 was achieved, and statistical analysis indicates that we can be 90% confident that the coverage exceeds 0.67. A further 12 events were analysed, of which four were ignored due to uncertainties in classification, four represented needless recovery, and the remaining four events were failures caused by the use of fault tolerance. If these four failures are offset against the 40 successes, then the notional coverage drops to 0.68.

The above analysis was guided by experience in running the command and control system in two modes: with and without fault tolerance. A direct comparison between these two modes of operation provided additional evidence of reliability enhancement. The Mean Time Between Failure for the fault tolerant software was 0.74 hours whereas without fault tolerance the MTBF was reduced to 0.31 hours (ratio 2.36). The proportion of missions completed without failure was 56% with fault tolerance enabled, compared with 47% when fault tolerance was not enabled (ratio 1.19).

Many of the failures of the fault tolerant system would not have occurred if reliable recovery mechanisms had been available, as would surely be the case if these techniques were to be used in practice for a succession of *different* application systems. If failures due to defective recovery are eliminated the experimental results indicate that a failure coverage of 90% and a nine-fold improvement in MTBF could be achieved. Almost 90% of missions would have been completed successfully.

Is design fault tolerance a cost-effective means of achieving reliability?

This final question is the most pertinent, but unfortunately a definite answer cannot as yet be given.

The results of the Newcastle project, summarised in the previous section, certainly indicate that design fault tolerance techniques can yield a significant increase in reliability. But was this improvement worthwhile? The reliability enhancement was achieved at a cost of: 60% extra in software development, 33% extra code summary, 35% extra data memory, and 40% run time overhead (largely due to additional synchronisation). These figures are probably on the high side, reflecting the novelty of the techniques, the extent of their utilisation, and the lack of fine tuning of the completed system. Furthermore, increased development costs can be offset by gains from economics in software testing.

However, the ability to engineer the reliability of a system is not so much a consequence of the availability of techniques for improving reliability as it is dependent on information concerning the relative cost-effectiveness of those techniques. In order to construct a system which will have a given level of reliability, within a fixed budget and adhering to project time-scales, the reliability engineer needs to select appropriate techniques and apportion the amount of effort to be devoted to each. Only when data is available on techniques of fault avoidance and removal as well as for fault tolerance will it be possible to make a rational determination of the best mix of reliability techniques. In the absence of such data (as is largely the case for software) I would argue for the eclectic approach. Optimal solutions are rarely achieved by putting all one's eggs in one basket. A well-engineered approach to building highly reliable software is likely to be based on striving for perfection, but at the same time recognising that imperfections will still be present - and therefore design fault tolerance will be needed to cope with them.

Conclusion

To provide a summary I reiterate my questions and answers.

Is there any need for design fault tolerance? Potentially yes, given our current inability to achieve perfection.

Is design fault tolerance a mature technology? Yes, in the sense that it is well developed and ripe for exploitation.

Can design fault tolerance be utilised in practical systems? Yes, this has been demonstrated.

Can design fault tolerance improve system reliability? Yes, experiments confirm that a substantial improvement can be achieved.

Is design fault tolerance a cost-effective means of achieving reliability? A firm maybe. This is the crucial question.

References

(1) ANDERSON, T. and LEE, P.A.: 'Fault Tolerance : Principles and Practice', Prentice Hall *International*, 1981.

(2) SLIVINSKI, T. et al.: 'Study of Fault Tolerant Software Technology', Report to NASA Langley Research Center, Mandex Inc., 1984.

(3) AVIZIENIS, A.: 'Design Diversity - The Challenge of the Eighties', *Digest of the 12th Int. Symp.* on Fault Tolerant Computing, Santa Monica, 1982, pp. 44-45.

(4) HORNING, J.J. et al.: 'A Program Structure for Error Detection and Recovery', in Lecture Notes in Computer Science 16, ed. E. Gelenbe and C. Kaiser, Springer-Verlag, 1974, pp 171-187.

(5) CHEN, L. and AVIZIENIS, A.: 'N-Version Programming : A Fault-Tolerance Approach to Reliability of Software Operation', Digest of the 8th Int. Conf. on Fault Tolerant Computing, Toulouse, 1978, pp. 3-9.

(6) LEE, P.A.: 'A Reconsideration of the Recovery Block Scheme', Computer Journal, 1978, 21 (4), pp. 306-310.

(7) SCOTT, R.K. et al.: 'Modelling Fault-Tolerant Software Reliability', Proc. of the 3rd Symp. on Reliability in Distributed Software and Database Systems, Clearwater Beach, 1983, pp. 15-27.

(8) SONERU, M.D.: 'A Methodology for the Design and Analysis of Fault-Tolerant Operating Systems', Ph.D. Dissertation, Illinois Institute of Technology, Chicago, 1981.

(9) ANDERSON, T. and KNIGHT, J.C.: 'A Framework for Software Fault Tolerance in Real-Time Systems', IEEE Trans. on Software Engineering, 1983, SE-9 (3), pp. 355-364.

(10) WELCH, H.O.: 'Distributed Recovery Block Performance in a Real-Time Control Loop', Proc. of *Real-Time Systems Symposium, Arlington*, 1983, pp. 268-276.

(11) CAMPBELL, R.H. et al.: 'Practical Fault Tolerant Software for Asynchronous Systems', 3rd IFAC/IFIP Workshop on Safety of Computer Control Systems, Cambridge, 1983, pp. 59-65.

(12) CRISTIAN, F.: 'Exception Handling and Software Fault Tolerance', IEEE Trans. on Computers, 1982, **C-31** (6), pp. 531-540.

(13) MARTIN, D.J.: 'Dissimilar Software in High Integrity Applications in Flight Controls', AGARD Symp. on Software for Avionics, The Hague, 1982, p. 36:1.

(14) WILLIAMS, J.F. et al.: 'Advanced Autopilot Flight Director System Computer Architecture for Boeing 737-300 Aircraft', 5th Digital Avionics Systems Conf., 1983.

(15) VON LINDE, O.B.: 'Computers can now Perform Vital Operations Safely', Railway Gazette International, 1979, **135** (11), pp. 1004-1006.

(16) GARMAN, J.R.: 'The "Bug" Heard 'Round the World', Software Engineering Notes, 1981, **6** (5), pp. 3-10.

(17) ANDERSON, T. et al.: 'An Evaluation of Software Fault Tolerance in a Practical System', Technical Report, University of Newcastle upon Tyne, 1985.

(18) ANDERSON T. and MOULDING, M.R.: 'Dialogues for Recovery Coordination in Concurrent Systems'. In preparation.

(19) KELLY, J. and AVIZIENIS, A.: 'A Specification Oriented Multi-version Software Experiment', Digest of the 13th Int. Symp. on Fault Tolerant Computing, Milan, 1983, pp. 120-126.

DEPENDABLE AVIONIC DATA TRANSMISSION

D. R. Powell J. C. Valadier

LABORATOIRE D'AUTOMATIQUE ET D'ANALYSE DES SYSTEMES DU C N R S
31077 TOULOUSE CEDEX - FRANCE

This paper outlines the major constraints imposed on the design of dependable local area networks for avionic systems and underlines the essential differences in requirements that exist with respect to those of ground-based (civil) LANs. The different choices available to the system designer are then discussed: technology (electrical or optical), architecture (bus or loop), general philosophy (centralized or decentralized), medium access control (competition or consultation). The paper then goes on to summarize two different and independent avionic LAN research projects, one which focusses on fault and damage-tolerance and the other on high-speed.

INTRODUCTION

Digital data transmission systems were in use in avionic applications before the term "local area networks" was coined. The aims that such systems were designed to satisfy were chiefly that of reducing aircraft wiring complexity and improving the ease with which the "integrated" avionic system could be extended. Such systems include ARINC 429 in the civil aviation domain and MIL-STD-1553 (USA) and GAM-T-101 (F) in the military area. The basic versions of these early avionic LANs enable the direct connection of at most 32 stations and operate at relatively low raw data rates (≤1 Mbit/s) over distances up to about 100 m. Moreover, the centralized request/response techniques used for controlling access to the medium lead to an effective data rate efficiency of less than 50% as well as a considerable lack of flexibility. As with all new technical offerings, the very existence of such LANs has led to the development of increasing numbers of new applications; user requirements have thus become more exacting: more stations, more flexibility, more bandwidth and longer distances to extend their applicability beyond the avionic field to, for instance, the marine area). Furthermore, user reliance on shared communication systems has increased the importance of making these systems dependable.

This paper discusses the design constraints imposed on embedded real-time local area networks (LANs) and the various options available to the system designer. Two different and independent research projects on dependable avionic LANs are then summarized:

- RHEA: a hierarchical LAN for aggressive environments in which the accent is placed on the tolerance of multiple faults due to physical damage,
- ANTINEA: a high-speed bus-topology LAN in which the emphasis is on high data rates and fast, dependable recovery techniques.

1. DESIGN CONSTRAINTS

Local area networks for harsh real-time applications and, in particular, avionic applications must satisfy a certain number of critical design constraints that are far more restrictive than those that dictate the design of LANs for office systems, mainframe back-end networks, industrial command-control systems, etc. The considerable amount of work that has been invested by LAN standardization committees such as IEEE802 cannot be directly extended to the harsh real-time, on-board environment due to the required real-time characteristics, the particularities of this environment and the restrictions necessary for practical utilization.

1.1. Real-time constraints

The fundamental difference between a real-time distributed system and other sorts of distributed systems is due to what we call the "real-time sandwich": both the highest and lowest layers of protocols interact with the real-world and must as such live with the basic rules of physics. The highest layer (the application layer) must interact with one or more physical processes whose dynamics are dictated by laws imposing absolute contraints on the communication delays and the maximum execution times of the information processing system. At the other end of the scale, the lowest layer (the physical layer) must send and receive signals over physical transmission channels which are again governed by a very basic physical law: the speed of light. This imposes a minimum delay on the transmission of a signal over the required distance (whatever the transmission data rate) which, in turn, fixes a minimum overhead on the time required to access the channel at the medium access control layer. The delay-performance restriction at each layer imposes delay restrictions on the next upper layer and so on right up to the application layer. The major problem of real-time distributed systems is to render all the protocol layers compatible with the physical constraints imposed on both sides of the sandwich.

In this paper, we are concerned only with the consequences of the real-time application on the design of the data transmission system or LAN (layers 1 and 2). One of the most obvious constraints on a real-time LAN is that message transfer times (and thus medium access times) be "bounded". If the minimum application time constraint is several orders of magnitude greater than the mean and the standard deviation of the message transfer time then it is sufficient to ensure a "stochastic" bound (such as that offered by a pure CSMA/CD (Ethernet) approach) for which the probability of this bound being exceeded is non-zero but considered negligible. If this is not true, then the medium access time must have a deterministic bound in the sense that, during normal fault-free operation, it is possible to prove that the application time constraints will be satisfied.

The application time constraints may be so demanding that certain application communication patterns may need to be organized as a single, uninterruptable transaction throughout which all resources must be

retained. At the LAN level, this leads to the notion of a command/response mode of message transmission whereby the response message must be transmitted immediately after the corresponding command message. The station issuing the command may require to further retain the transmission resources to transmit a confirm or commit message.

In order to synchronize several (real-time) actions between two or more tasks (such as inputting time correlated data), real-time constraints may impose the provision of a single, physical broadcast transaction. Furthermore, the updating of system state information may need to be carried out periodically with a low jitter around the nominal update period. This leads to the requirement that the LAN be capable of transferring a certain number of periodic_messages.

Lastly but not least, most real, real-time systems are priority driven: tasks may be assigned different fixed priorities or their priorities may evolve dynamically in order to meet application deadlines. This task priority feature in turn leads to the requirement that the LAN be capable of implementing message_priority_classes. Preferably, the message priority implementation should enable maximum utilization of LAN bandwidth by low-priority messages if no high-priority messages are outstanding and should minimize the time required for a high-priority message to preempt a low priority message stream.

1.2. Environmental constraints

Real-time LANs must often operate in demanding and even aggressive environments; this leads to quite severe constraints on the transmission technology and LAN architecture.

One of the most obvious constraints is that of electromagnetic compatibility. The real-time LAN environment is full of such nasty things as electric arcs, high-power radio and radar transmissions and, in the military area, electronic counter-measures, electromagnetic pulses and radiation. The technology chosen for the transmission medium and for the electronic equipment must have a high_immunity_to_interference of this kind.

Other, less obvious constraints imposed by the environment include: resistance to moisture, high temperature, fire, chemical spilling, rats, cleaning personnel, missiles, etc. Depending on the nature of the environment, the LAN might have to tolerate_physical_damage possibly resulting in multiple, common-mode faults.

Of particular importance in the case of avionic systems is that the available on-board electrical power is limited. During normal operation, load-shedding may occur whereby non-critical equipment is momentarily disconnected. In the case of military systems, some equipment (weapons) may be physically ejected from the system. The consequence on the LAN is that its design must take account of multiple_station_removal in the course of normal_operation. The communication service offered to the other stations should not be unduly degraded on occurrence of these normal events.

1.3. Exploitation constraints

One of the objectives of any LAN, real-time or not, is to provide system flexibility and evolutivity. This means that it must be relatively simple to remove and to add stations to the LAN. In the case of a real-time LAN, the required flexibility is aggravated by several aspects. Firstly, the addition of new functions may involve the installation and test of several new pieces of equipment with complex synchronization patterns. The system integrator will prefer to carry out the installation incrementally; the LAN must thus be able to function in spite of the absence_of_stations that have not yet been tested or that have been disconnected to allow diagnosis of the new function. Secondly, system diagnosis for installation and maintenance operations is conceptually easier if it can be carried out from a single point; centralized_modes of LAN operation and LAN monitoring would advantageously simplify these operations. Thirdly, access to certain parts of the environment (particularly in the case of aircraft) can be difficult and costly; alterations to the transmission cables may be impossible or beyond the scope of the system integrator's responsibilities. Consequently, it is preferable to envisage a pre-wired_system in which stations can be easily connected at any point and in an arbitrary order.

Furthermore, the applications of real-time LANs almost always require a high_system_dependability. Service interruption may not only lead to user "inconvenience" but could be catastrophic in terms of human lives, cost, production or mission effectiveness. The LAN must thus itself be dependable.

2. TECHNOLOGY: ELECTRIC OR OPTIC ?

The choice between the use of electrical cables or optical fibers for physical signal transmission within the LAN is important and can not only dimension the physical size and speed of the system and its immunity to electromagnetic interference (d'Hervilly 84) but also strongly impacts the possible architectural choices.

2.1. Electric cable technology

The skin effect in electrical cables leads to losses that are proportional to the length of the cable (L) and to the square-root of the transmission frequency (F). This means that the maximum achievable transmission distance with a given type of cable is inversely proportional to the square-root of the maximum transmission frequency. The actual maximum product $L.F^{1/2}$ for a particular cable is itself approximately proportional to the cable diameter.

In practice, for transmission rates of 30 to 50 Mbit/s, small-dimension cables (3 to 4 mm. diameter co-ax) can be used over distances up to about 50 meters (aircraft). In terrestrial or marine applications, the weight of larger-diameter cables (1 to 1.5 cm.) is no longer a limiting factor and such cables can be used to cover distances of 200 to 300 meters. If longer distances need to be covered then signal-repeaters

become necessary.

As regards immunity to electromagnetic interference, the utilization of electric cables requires the selection of co-axial or tri-axial cables together with compatible connectors. Electric cable systems have been implemented that are successfully immune to the severity of electro-magnetic pulse interference.

2.2. Fiber optic technology

The speed limitations of a fiber-optic link depend not only on the characteristics of the fiber but also on the transmitting and receiving components. The temperature ranges over which avionic LANs are required to operate impose the use of LED transmitters and PIN-diode receivers. Even though optical fiber is capable of transmitting at very high speed, this restriction of terminal devices in practice limits the maximum transmission rate to about 50 to 100 Mbit/s. With a judicious choice of wavelength and fiber (1300 nm., graded-index fiber with 100 micron core), links up to several kilometers can be achieved at the envisaged transmission rates.

However, if it is necessary (as in combat aircraft) to implement field-disconnectable sections of optical cable then the losses introduced by the connectors (negligible in electrical systems) can greatly decrease the maximum achievable transmission distance.

Of course, from the viewpoint of immunity to electromagnetic interference, optical fiber presents a distinct advantage over electrical cable (especially in the future when the use of composite non-metallic materials will cease to offer the Faraday-cage effect of present-day aircraft). Unfortunately, the very low signal levels at which the receiving devices must often operate implies that they constitute the Achilles heel of fiber-optic links with respect to noise immunity.

With today's state-of-the-art and with the presently required transmission rates and distances (in aircraft), fiber-optic technology does not offer a distinct advantage over electrical cable. A wise decision for the design of an avionic LAN would be to make it compatible with the use of either technology (or a mix of both).

3. ARCHITECTURE: BUS OR LOOP ?

The two basic architectures for the realization of LANs are the bus and the loop (other architectures such as packet or circuit-switched meshed networks have been proposed but suffer from the fact that single-point monitoring of data transmissions is not possible). Common to both architectures is the problem of appropriately choosing the protocol that stations must obey in order to access the shared medium. In this paragraph, we discuss the specificities of these two basic architectures from a transmission technology viewpoint.

3.1. Bus architecture

In a bus architecture, stations are tapped onto a multi-point physical transmission medium. In its basic form, this physical medium is passive but can be extended by means of repeaters or arbiter/repeaters. Each station is in one of two states: the receiving state or the transmitting state. When a station successfully acquires the bus (according to the appropriate access protocol), all the other stations are able to simultaneously receive the transmitted signal (neglecting propagation delays). The overhead in terms of preamble length to ensure receiver phase-locking is thus independent of the number of connected stations; it is thus feasible (and usual) to carry out receiver clock synchronization at the beginning of each message.

The main technological design problem in bus-architectures is that of configuring the transmission medium and station taps in order to maximize the allowable number of connection points and to minimize the effects of faults. In its simplest form, an electrical bus consists of a linear cable onto which stations are tapped by T-connectors. Each T-connector introduces a cable mismatch; this introduces limitations on the maximum number of connections and their relative positions. In order to remove restrictions on the length of the stubs between the cable and the stations, active T-connectors can be envisaged but this leads to a less flexible implementation in that active elements must be embedded into the aircraft's infrastructure. One very interesting technique that alleviates most of these problems is to use CATV-style directional coupling onto a unidirectional cable. This doubles the amount of cable in the system but offers considerable advantages:

- the taps are entirely passive,
- attenuation on the stubs is the only limiting factor as regards their length,
- the bus can incorporate passive branches into damage-prone areas,
- impedance mismatches due to cable or stub faults (open or short circuits) are not necessarily catastrophic (depending on their location).

The use of fiber optic technology in a bus architecture leads to another set of problems that are related to the attenuation introduced by optical coupling and splitting components. Linear optical bus topologies are very quickly limited by the attenuation introduced by bidirectional optical T-couplers and it is necessary to resort to either a star topology or a configuration using directional couplers similar in principle to the CATV electrical technology mentioned above.

3.2. Loop architecture

In the basic loop architecture, each station is connected to two other stations by point-to-point transmission links in order to form a ring. Signals are transmitted around the ring unidirectionally. Each station may be in one of two states: the transmitting state in which it sends its own signal onto the downstream link or the repeating state in which it relays signals received on the upstream link. In both states, a station can decode signals received on the upstream link. One of the claimed basic advantages of the loop over the bus is that signals are actively relayed; this means that the maximum transmission

distance and the number of connected stations can be much greater than for a bus. It also means that physical links are point-to-point and thus better adapted to fiber-optic technology. However, this active signal regeneration leads to several disadvantages.

The first disadvantage is of course that the station attachment units must be powered. If power is obtained locally from the connected station then the loss of a station implies that the loop is broken and no further communication is possible. Other than independently powering the signal relaying circuitry, several techniques (or a combination thereof) are possible in order to re-establish loop continuity after station disconnection:

- station bypassing,
- braided loop,
- twin counter-rotating loops,
- bidirectional loop.

Station bypassing is the most basic technique [Penney 74]: when a station is disconnected the incoming signal is passively relayed onto the outgoing link. Fiber-optic implementations of passive bypassing are possible but the number of consecutively bypassed stations is quickly limited due to the extra attenuation that is introduced. A related technique, braiding, achieves bypassing by providing extra links that interconnect non-adjacent stations [Ambrus 85]; here again, the number of consecutively bypassed stations is limited by the jump interval of the braiding links. The counter-rotating loop approach [Zafiropulo 74] enables a single loop to be configured after the failure of a single station (or a pair of inter-station links); the loss of several non-consecutive stations leads to partitioning of the loop. Also, a pre-requesite to loop reconfiguration is fault detection and localization by means of a possibly time-consuming distributed diagnostic routine. The bidirectional loop approach [Wolf 79] relies on the use of an "oscillating" token access protocol when the loop is no longer continuous; each time a station passes on the token, it inverts the direction of its signal relaying device: this basically means that what remains of the loop is used more-or-less as an active bus with rather a long propagation delay, thus losing the performance advantage of the original loop topology.

A second disadvantage stems from the fact that in order to relay incoming signals, the station attachment unit must first lock its receive clock onto the incoming signal. If stations are required to relock their receive clocks at the beginning of each message then two implementations can be considered. If the delay in each station is restricted to a single bit (as in IEEE 802.5) then the required preamble overhead is proportional to the number of connected stations since several bits of preamble are lost in each station due to the required number of bits necessary for their receive clocks to resynchronize. If a constant length preamble is regenerated in each station then the repeat delay in each station must cater for the time required for the receive clock to resynchronize [Bean 85]. In both implementations, there is a considerable delay degradation due to the serial disposition of the stations in the loop.

As a consequence, loop architectures generally operate in a bit-synchronous fashion at the physical level; whether transmitting or repeating, each station (except one) is permanently locked onto the phase of the incoming signal. One of the stations (called the active monitor in the IEEE 802.5 proposition) assumes the priviledged role of generating the master clock signal. In the event of station disconnection or link failure, no useful communication is possible until every station's receive clock has been resynchronized.

If the active monitor should be disconnected (e.g. due to load-shedding) then the duration of communication service interruption is even longer since a new monitor must be elected among the potential monitors. In IEEE 802.5, the potential monitors are informed of the active monitor's health by means of periodic poll messages. The time-out value necessary for a potential monitor to detect that the active monitor has died must be greater than period of these poll messages; unfortunately, this period cannot be too short or the loop would be overloaded with polling messages. In IEEE 802.5, the default poll period is 3 seconds and the default detection time-out is set as 7 seconds. Service interruptions of this magnitude are totally unacceptable in a harsh real-time environment.

To summarize, the major reasons for which we exclude loop architectures are:

- the inherent difficulty of ensuring correct operation when several stations are absent (during system integration or following station disconnection due to failure or load-shedding),
- the excessive duration (in harsh real-time applications) of the communication blackout that results when the loop must be reconfigured and resynchronized.

We should emphasize that, in our opinion, these reasons for rejecting loops outweigh the advantages that the basic fault-free loop architecture offers with respect to performance (insensitivity to maximum system dimension) and the implementation of preemptive access priorities (by bit-flipping reservation techniques).

4. PHILOSOPHY: CENTRALIZED OR DECENTRALIZED ?

4.1. The centralization/decentralization dichotomy

When investigating the properties of distributed systems with respect to dependability, two conflicting viewpoints can be considered:

- the optimistic viewpoint: if all functions are totally decentralized than redundancy is maximized since the failure of any single entity cannot lead to total system failure,
- the pessimistic viewpoint: the more that functions are decentralized, then the higher the number of entities involved and the higher the chances that the failure of one of them will lead to total system failure.

The key to this dichotomy is the assumption that is made with respect to the consequences of a fault in one of the entities. If one assumes that when an entity fails that it does so in a "nice" way whereby errors are not propagated towards the other entities then the optimistic viewpoint is the correct one.

Unfortunately, such an assumption of "nice" failures is rather difficult to justify. On the other hand, to predict that an entity failure will bring down the system is a far too gloomy approach. These diverging viewpoints can be quantified mathematically by the notion of coverage which is defined as the probability that errors are detected and confined, conditioned on the fact that a fault has occurred. The question as to which of the above viewpoints wins out can only be answered in function of the assumed coverage values.

In a shared medium communication system (loop or bus), one of the essential system-wide functions is that of controlling access to the medium; in this section we are concerned by the degree to which this function should be decentralized.

4.2. Influence of error confinement coverage on path reliability

Two sorts of LAN stations are considered:
- master stations: at least one master station (the active master) must be operational if the system is to work correctly; if the active master fails then a new active master must be elected from the other potential master stations,
- slave stations: the (confined) failure of a slave station does not lead to system failure; a slave station is different from a potential master station in that the logic required to become an active master is either omitted or physically inactivated.

Totally centralized and totally decentralized systems are respectively those in which only one station, or all stations, can assume the role of master.

We define the fault-coverage of master and slave stations as p_m and p_s respectively and make the important assumption that p_m is less than p_s; the reasoning behind this assumption is that when a master station (active or potential) fails, the probability $(1-p_m)$ that it will do so in such a way as to prevent system recovery is higher than that of a slave station because the enabled logic in the latter is less complex than the former. The non-perfect coverage means that there is point beyond which the additional probability of system failures due to an extra master station outweighs the fact that there is an additional spare [Stiffler 78].

Let N be the total number of stations (master or slave), m be the number of master stations and R be the station reliability (assumed equal for master or slave). Disregarding failures of the physical medium, the path reliability R_{ss} between a pair of slave stations is given by:

$$R_{ss} = \underbrace{R^2}_{\Pr(\text{both stations OK})} \cdot \underbrace{\{ [R + p_m.(1-R)]^m - (1-R)^m \}}_{\Pr(\text{at least one master OK and others either OK or failed with correct error confinement})} \cdot \underbrace{[R + p_s.(1-R)]^{N-m-2}}_{\Pr(\text{other slaves OK or failed with correct error confinement})}$$

Similarly, the path reliabilities between a pair of master stations R_{mm} or a master station and a slave R_{ms} are given by:

$$R_{mm} = R^2 \cdot [R + p_m.(1-R)]^{m-2} \cdot [R + p_s.(1-R)]^{N-m}$$

$$R_{ms} = R^2 \cdot [R + p_m.(1-R)]^{m-1} \cdot [R + p_s.(1-R)]^{N-m-1}$$

Thus, the mean path reliability RP between an arbitrary pair of stations can be expressed as follows:

$$RP = \frac{1}{N(N-1)} \cdot [m(m-1).R_{mm} + 2m(N-m).R_{ms} + (N-m)(N-m-1).R_{ss}]$$

Figure 1 shows the path un-reliability (1-RP) plotted as a function of $(1-p_m/p_s)$ for fixed values of p_s, R and N and for various values of m. For p_m greater than 0.9996 (left of point A), the path reliability is insensitive to the number of master stations as long as there are at least two. As p_m decreases (right of point A), the totally-decentralized solution (m=50) becomes worse than the partially-decentralized solutions and at point B (corresponding to p_m=0.9799) it becomes worse than even the fully centralized solution. Space prevents us from showing other curves for different parameter values but it should be noted that low values of R (i.e. long mission times) favor the fully decentralized approach.

In conclusion to this discussion on decentralization versus centralization, in systems with short mission times or where repair is possible, the extra redundancy brought about by a high degree of decentralization may not only be unnecessary but in some cases harmful to overall system dependability.

5. MEDIUM ACCESS CONTROL: COMPETITION OR CONSULTATION ?

Medium access control (MAC) protocols for both busses and loops may be classed into two categories:
- consultation protocols: stations consult each other by means of special-purpose messages or sub-frames in order to decide which station has the right to transmit; the "right-to-transmit" is exchanged in an orderly, synchronized fashion in order to avoid conflicts,

- competition_protocols: stations that wish to send a data message compete for the right to transmit; the "right-to-transmit" is asychronously created by the winning station. conflicts are unavoidable but tolerated.

In the case of bus architectures, the archetypes of these two categories are respectively the token-passing and CSMA/CD protocols defined by the IEEE 802.4 and 802.3 sub committees.

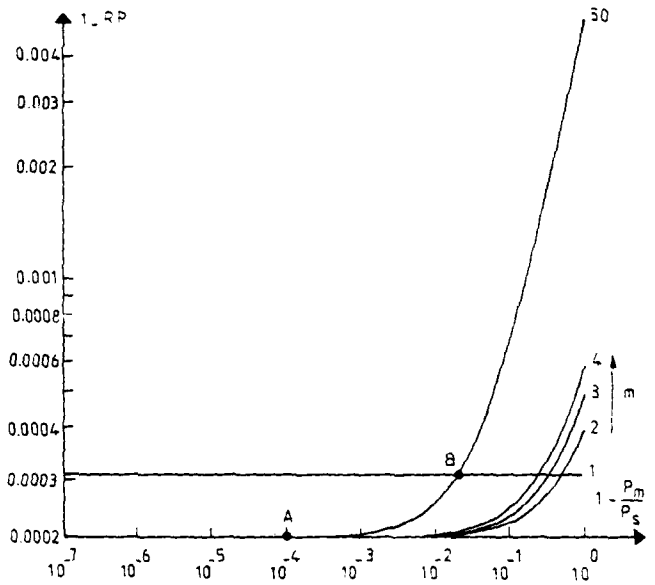


Figure 1: Mean pairwise path un-reliability
(N=50, R=0.9999, p_1=0.9999)

From a dependability viewpoint, competition protocols would seem to be preferable since conflicts are natural foreseen events whereas in consultation protocols, if *stations* become desynchronized then unexpected conflicts *may* occur and special mechanisms must be invoked in order for the stations to resynchronize and resume normal operation. These special, rarely-activated mechanisms lead to two problems. Firstly, if desynchronization should occur, the effective access time may be much greater than the one expected. Secondly, the irregular invocation of the resynchronization mechanisms leads to an error-latency problem which could lead to the presence of multiple, undetected faults from which the system may be unable to *recover*. This viewpoint regarding the impact of MAC protocols on system dependability, which we cannot as yet justify either mathematically or experimentally, leads us to prefer the use of competition protocols.

5.1 Bounded CSMA protocols

The main disadvantage of competition protocols in their basic *CSMA/CD* form is the fact that medium access delays possess no *deterministic* upper bound. Although a "stochastic" upper bound may be sufficient in some applications (see paragraph 1.1) such an assumption cannot be justified in avionic systems where the time constraints are in the order of milli-seconds. In order *to deterministically* bound access times, it is necessary *to be able* to dynamically compute and implement access priorities between competing stations [Valadier 84]. To explain how priorities can be introduced into CSMA protocols, suppose that the instantaneous priority of a message is equal to p=1..P, where P is the maximum priority and that a slot is a period of time greater than or equal to twice the end-to-end propagation time on the bus plus the time necessary for a station to detect a carrier on a previously free bus or to detect a conflicting signal (the latter are assumed equal). There are three basic techniques for implementing priorities:

- deference_delays: each message is preceded by a delay whose length D in slots is a linearly decreasing function of its priority level: D(p)=P-p; *messages* with low priorities will detect the carrier *due to a* higher priority message and continue to defer (this implementation only avoids conflicts if the bus never goes free for more than P slots),
- preamble_lengths: each message is preceded by a *preamble* whose length L in slots is a linearly increasing function of its priority level: L(p)=p; messages with low priorities will detect conflicts due to the longer preamble of a higher priority message and cease to transmit,

- forcing_headers: the transmission channel must effectively carry out a logical "or" of transmitted signals and each message is preceded by a header of fixed duration H slots consisting of a "start-bit" and the binary representation of the message's priority (MSB first) P(p)=p; in a particular slot, stations sending logical "0" will stop transmitting if they detect a "1" (the actual representation of a "1" need not be a continuous level but could be a burst of carrier).

In all of these techniques, there is an overhead for each message that is an increasing function of both the slot-duration (and thus the maximum length of the bus) and the maximum number of priority levels, P. Of the three, the forcing header implementation gives the lowest header overhead (for P>6) since its length increases only as the log of P.

However, with the forcing header implementation, there is a further contribution to the overall overhead due to the fact that an extra "trailer" time must be allowed for after a message has been transmitted. This is due to the fact that when the active message priority p is zero, the interval between the start bit and the beginning of the data part of the message is effectively perceived by the listening stations as an idle time (bus unoccupied); thus, a station must wait for at least an equal interval after the end of message transmission before it can say for certain that the bus has become free. The overall overhead would thus be twice that due to the header if no further precautions were taken. In fact, the overall overhead can be minimized by inserting a fixed number of "1" bits into the header so as to decrease the maximum bus-idle time when the active message priority is zero.

Using the following notation:

ceil(x) : smallest integer greater than x
K : number of bits needed to code the message priority:
b : number of extra "1" bits inserted in the header (overhead optimization bits)
TH : header duration in slots
TT : trailer time duration in slots

we obtain:

$$K = \text{ceil}\ [\ \log_2(P)\ ]$$
$$TH = 1+K+b$$
$$TT = \text{ceil}\ [\ K/(b+1)\ ]$$

and the overall overhead (TH+TT) is minimized for:

$$b = \text{ceil}\ [\ K^{1/2} - 1\ ]$$

We have studied two bounded access-time CSMA protocols based on the forcing header concept that we shall denote CSMA/FIFO and CSMA/VR.

5.1.1 The CSMA/FIFO protocol

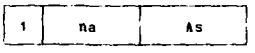
In this protocol the instantaneous priority of a message is given by:

$$p = N.na + As$$

with:

N : the maximum allowed number of stations (power of 2)
na : the number of failed access attempts incurred by the message
As : the source address of the message (As=0..[N-1])

The forcing header is thus made up as follows (neglecting the overhead optimization bits):



Thus, messages with highest values of na will have priority in gaining access to the bus and messages with equal values of na will be seperated according to their source address. Whenever a station fails to send a message (because another station has a higher instantaneous priority), it waits until the successful station has finished sending and then tries again immediately. This access protocol thus has the effect of providing a FIFO-like scheduling technique for the bus.

The maximum access time that is achieved occurs for station "0" when it generates a message just after all the other stations collide in attempting to access the channel; it must thus wait for a total of n message transmissions before it can successfully transmit (where n is the number of active stations). Letting TD be the duration (in slots) of the data part of a message and allowing 1 slot between each successive message (the worst case), the maximum access time (in slots) is given by:

$$TAmax = n.(TH+TD+TT+1) + TH$$

With N=128 and b=3 this protocol leads to:

- TH = 18 slots
- TT = 4 slots
- TAmax = [n.(TD + 23) + 18] slots

5.1.2 The CSMA/WR protocol

This protocol is based on what we call a "waiting-room" concept: each station may insert at most one message in a virtual waiting-room of messages competing for the medium and can only do so when the waiting-room is empty. The status of a message with respect to the waiting-room is indicated by a local boolean variable W (0: outside, 1: inside) and the status of the waiting-room is indicated by a global boolean variable W (0: empty, 1: non-empty). A newly generated message has a zero W-value; the latter may only be set to one when the waiting room is non-empty. The instantaneous priority of a message from a station of address As is set equal to:

p=W.N+As

The forcing header (neglecting the overhead optimization bits) thus consists of the start-bit, the W-bit and log2(N) address bits:

When a conflict occurs, the value read by each station during the W-bit slot is none other than the global variable W. If a station is attempting to send a message with W=0 then it will stop sending if the value of W is equal to 1 (indicating that the waiting-room is non-empty). Messages in the waiting-room acquire access in an orderly and thus bounded fashion by means of the fixed priority order imposed by their address As. When the last message in the waiting-room has been sent, W will be equal to 0 and all outstanding messages (with W=0) will collide and consequently set the W-values to 1 (indicating that the waiting room let in a new set of messages).

The maximum access time that is achieved occurs for station "0" when it generates a message just after all the other stations enter the virtual waiting room and then immediately generate new messages as soon as they have successfully transmitted; station "0" must thus wait for a total of 2.(n-1) message transmissions before it can successfully transmit. The maximum access time (in slots) is thus given by:

$$TAmax = 2.(n-1).(TH+TD+TT+1) + TH$$

With N=128 and b=2 this protocol leads to:

- TH = 11 slots
- TT = 3 slots
- TAmax = (2.(n-1).(TD + 15) + 11) slots

5.1.3 Performance limits

The table in figure 2 summarizes the overheads incurred, the corresponding maximum access time and the frame and protocol efficiencies for raw data rates of 1 and 40 Mbit/s. The frame efficiency is defined by the ratio between the useful data duration and the overall message duration and the protocol efficiency is defined as the ratio between the achieved maximum access time to the one that would be achieved by a perfect FIFO scheduling scheme with zero overhead (i.e. (n-1).TD).

	1 Mbit/s		40 Mbit/s	
	CSMA/FIFO	CSMA/WR	CSMA/FIFO	CSMA/WR
Message overhead (µs)	66	42	66	42
Max access time (ms)	20.9	38.0	4.88	6.51
Frame efficiency (%)	79.5	85.9	8.84	13.2
Protocol efficiency (%)	77.3	42.5	8.26	6.19

Figure 2: Bounded CSMA protocol comparison
(with 256-bit messages, a 300 m. bus, N=128 and n=64)

It can be seen that although the CSMA/WR protocol gives a better frame efficiency than that of the CSMA/FIFO protocol, the fact that worst-case CSMA/WR messages must wait for a higher number of higher priority messages leads to a protocol efficiency almost half that given by CSMA/FIFO. In both cases, when high raw data rates are employed (meaning that the slot duration is no longer negligible when compared to the useful data part of the message), both protocols lead to a prohibitively low efficiency. We must thus conclude that such protocols can only be used when the data rate is low and/or bus lengths are short. The only way to achieve a higher performance on the bus architecture is to increase the degree of synchronization between the stations, i.e. to use a consultation protocol. We thus turn our attention to token bus protocols.

5.2. Token-passing protocols

The token bus protocol, in the fault-free situation, is extremely simple: a token is passed around a virtual ring of stations and only the station possessing the token has the right to initiate transmissions. From a performance viewpoint, the overhead per information message is fixed by the duration of the token message. Assuming a token message of 8 octets, the table of figure 3 gives the overheads incurred the corresponding maximum access time and the frame and protocol efficiencies for raw data rates

of 1 and 40 Mbit/s (note that the frame overhead and efficiency refer only to the time taken to transmit the token message and not the control information in the data message).

	1 Mbit/s	40 Mbit/s
Message overhead (µs)	21.3	1.6
Max access time (ms)	20.4	0.695
Frame efficiency (%)	80.0	80.0
Protocol efficiency (%)	79.1	58.1

Figure 3: Token bus performance
(with 256-bit messages, a 300 m. bus, N=128 and n=64)

It can be seen that at a data rate of 40 Mbit/s, there is a very distinct improvement in performance over the bounded CSMA protocols studied previously (see figure 2).

Another advantage of token passing is that it is well-adapted to the notion of multiple message transactions whereby the station holding the token may carry out a command/reponse sequence (see paragraph 1.1).

5.2.1 Operational continuity with token bus protocols

However simple token-passing protocols may seem in the fault-free situation, considerable complexity is introduced by virtue of the mechanisms necessary to initialize and maintain the virtual ring and to regenerate lost tokens. When the token is lost or stations are withdrawn, the communication service offered to the remaining stations is disrupted while recovery takes place. Unfortunately, most token-bus protocols are optimized for recovering the virtual ring in *single-error situations such as loss of* the token or of a single station (a quite reasonable assumption if ring integrity is only menaced by random physical faults). In a multiple error situation, all the protocols that we know of must totally re-initialize the virtual ring (by techniques akin to priority CSMA protocols) leading to quite lengthy service disruptions.

The situation is particularly delicate in the case of high-speed transmission in an avionic environment. Firstly, electromagnetic interference can lead to bursts of noise that will entirely obliterate whole sequences of messages resulting in the need for complete ring re-initialization. Secondly, even if long bursts of noise can be prevented, the normal avionic event of load shedding (see paragraph 1.2) can lead to the simultaneous withdrawal (and later re-insertion) of several stations.

We shall consider the duration of the resulting service interruption for two well-known token-bus protocols: ARCNET and IEEE 802.4. We shall restrict our comparison to the case where n stations are simultaneously withdrawn from the system, including the one that had the token. In order to compare the principles and not the technological implementations, we shall consider for both a maximum of 256 stations (8-bit address fields) and use the following notation:

- As : a station identity number (As=0..255)
- T : length of control messages (assumed equal for simplicity)
- slot : twice the end-to-end propagation delay
- n : number of active stations remaining after the incident leading to *ring re-initialization*

In both protocols, all stations detect the absence of activity on the bus by means of a time-out and *then* proceed to re establish the logical ring in two phases:

Phase 1: elect a station to generate a unique token
Phase 2: build the logical ring starting from station elected during phase 1

ARCNET Scenario

Phase 1: Each station starts a timeout equal to 2 T (255-As). In the worst-case, As is equal to *n*
Phase 2: The station that is first to time out polls the other stations starting from its As until it obtains a positive reply before a time-out equal to at least 1 slot; the replying station then starts polling from its address. This is repeated until the last active station polls the station that initiated phase 2.

IEEE 802.4 Scenario

Phase 1: All stations attempt to claim the token by sending "claim_token" messages; conflicts between claiming stations are iteratively resolved by using variable-length information fields that are multiples of the slot duration determined by successive pairs of bits in its As (*in a similar* fashion to CSMA protocols with priorities implemented by variable-length preambles).
Phase 2: The station that wins in phase 1 then sends a "sollicit_successor" message that thus initiates a "resolve_contention" sequence; contentions are resolved iteratively by requiring the other stations to reply after a number of slots that is again function of successive pairs of bits in their As. The station that wins then initiates another sollicit_successor/resolve contention sequence, etc.

Thus, in both protocols the duration of the service interruption (defined as the interval from the occurrence of incident leading to system re-initialization to the instant when the logical ring has been

re-established) can be calculated as a function of the required number of control messages (M) and slots (S) required to complete phases 1 and 2 (see table of figure 4).

	IEEE802.4						ARCNET					
	Phase 1		Phase 2		Total		Phase 1		Phase 2		Total	
n	M	S	M	S	M	S	M	S	M	S	M	S
2	4	7	14	19	18	26	0	254	258	255	258	509
4	4	7	36	50	40	57	0	252	260	255	260	507
8	4	7	78	108	82	115	0	248	264	255	264	503
16	4	7	162	212	166	219	0	240	272	255	272	495
32	4	7	328	416	332	423	0	224	288	255	288	479
64	4	7	660	776	664	783	0	192	320	255	320	447
128	4	7	1322	1492	1326	1499	0	128	384	255	384	383
256	4	7	2646	2730	2650	2737	0	0	512	255	512	255

Figure 4: Ring initialization in terms of control messages and slots (assuming 8-bit addresses)

The table in figure 5 gives the same comparison in terms of time (in milli-seconds) for a 300 m. bus with data rates of 1 and 40 Mbits/s and assuming that all control messages contain 8 octets. Figure 5 also shows the maximum access time that is ensured in the fault-free situation when 256-bit data messages are assumed.

	1 Mbit/s			40 Mbit/s		
n	Init. time (ms)		Fault-free access time (ms)	Init. time (ms)		Fault-free access time (ms)
	IEEE802.4	ARCNET		IEEE802.4	ARCNET	
2	0.258	4.107	0.387	0.082	1.591	0.0126
4	0.571	4.121	1.03	0.181	1.586	0.0346
8	1.165	4.149	2.33	0.365	1.575	0.0786
16	2.317	4.205	4.91	0.698	1.553	0.167
32	4.589	4.317	10.1	1.352	1.509	0.343
64	8.989	4.541	20.4	2.515	1.421	0.695
128	17.757	4.989	41.1	4.828	1.245	1.40
256	34.711	5.885	82.4	8.873	0.893	2.81

Figure 5: Ring initialization times and fault-free access times
(300 m. bus, 8-bit addresses, 8-octet control messages, 256-bit data messages)

This table leads to several comments:

1) With 8-bit addresses, when more than 32 stations are present in the virtual ring, the ARCNET initialization scheme is faster than the IEEE 802.4 scheme; note however that 16 or 48-bit addresses (as specified in the IEEE 802 standards but un-necessary in avionic systems) would severely penalize the ARCNET approach.

2) At a raw data rate of 1 Mbit/s, both initialization techniques lead to a service disruption that is less than the normal, fault-free maximum access time (for n>16 with the ARCNET approach).

3) At a raw data rate of 40 Mbit/s, the service disruption due to initialization using the IEEE 802.4 technique increases with the number of active stations and is from 4 to 8 times longer than the fault-free maximum access time. The ARCNET technique gives a service disruption that decreases with the number of active stations (the main delay is due to the phase 1 time-out) but varies from 126 down to 0.31 times the fault-free maximum access time.

In conclusion, the service disruption that results from the requirement for totally re-initializing the virtual ring when several stations are simultaneously disconnected due to the normal avionic event of load-shedding becomes prohibitively long when the raw data rate is high. In order to decrease the duration of the service disruption, the membership status of the virtual ring must not be lost when load-shedding occurs.

5.2.2 Partially-centralized recovery

Memorization of the status of the virtual ring can be achieved in two ways:
the logical ring can be permanently stored in non-volatile memory; this approach is very restrictive from the modularity viewpoint since the addition of new stations would require all station interfaces to be updated,
- certain priviledged stations can be equipped with back-up power supplies such that they are never powered down during load-shedding; these stations can then store the status of the logical ring and take responsibility for ring recovery.

The latter approach is quite feasible since certain highly-critical stations (such as inertial

navigational systems) must be continuously powered and will only withdraw from the system in the event of failure (a rare, rather than a normal, event such that multiple withdrawals of such stations can again be considered of negligible probability).

Thus, a variation on the basic token-passing scheme can be considered whereby only a subset of stations (master stations) take the responsibility for re-initializing the virtual ring, the other (slave) stations are physically prohibited from attempting ring recovery. One of the master stations assumes the role of an active_monitor (the other master stations assume the role of potential_monitors) that continuously follows the whereabouts of the token. If the token is lost or sent to a station that has withdrawn, the active monitor sequentially polls the successors of the last station seen to have had the token (not all possible stations, only those that were inserted in the virtual ring). It can thus be ensured that the communication service is rapidly restored to the remaining stations. In fact, the duration of the service disruption following a load-shedding operation is almost negligible and is proportional to the number of missing stations (not the number of active stations). For each missing station, there are the following contributions to the incurred service disruption:

- a one-slot time-out for the active monitor to realize that recovery must be initiated,
- a poll message to the station that should have transmitted (in case it did not reply because the token message was garbled),
- another one-slot time-out to detect that the station is indeed missing.

The duration of the service disruption can thus be considered equal to two slots per missing station since the poll message is of the same duration as the token message that would have been sent had the station not been disconnected.

As shown in paragraph 4, this apparent centralization of the recovery function is in no way detrimental to the overall system dependability (assuming that there are at least two or three master stations) and, depending on the relative fault coverages of the master and the slave stations, may be even better than a totally-decentralized approach.

Of course, if the station that is the current active monitor fails then a new active monitor must be elected in much the same way as in a conventional token-bus initialization or recovery protocol.

—◆—

In the preceding sections of this paper, we have outlined the major options that are available for real-time LANs in an avionic environment. In the next two sections, we shall give short descriptions of two research projects of avionic LANs in which we have been involved.

—◆—

6. RHEA: A HIERARCHICAL LAN FOR AGGRESSIVE ENVIRONMENTS

The avionic LAN summarized in this section was designed in collaboration with the Crouzet company. RHEA was the subject of a prospective research project and as such its specifications were very general. The emphasis in its design was placed on fault and damage-tolerance and from a performance viewpoint, the objective was to be comparable with existing avionic data transmission systems, i.e. from 100 kbit/s to 1 Mbit/s.

6.1 Network-level architecture

In a bus architecture, the only element common to all inter-station paths is the bus itself. Consequently, we would expect such an architecture to be very dependable if the multipoint channel is correctly designed. This qualitative reasoning was confirmed by a quantitative comparison of various transfer architectures based on a simulation approach (Powell 82). In the case of an aggressive environment where physical damage must be taken into account, this same study revealed that meshed architectures such as a braided loop or an irregular network were more apt to tolerate multiple localized faults (lesional faults) due to damage. The philosophy adopted in RHEA was to try to obtain the simplicity advantages of the bus architecture whilst relegating the responsability for lesional fault tolerance to the physical level. However, a requirement for interconnecting geographically localized stations into clusters led us to choose a derivative of this architecture possessing two levels of multipoint channels (figure 6):

- a distributed bus interconnecting geographically distributed stations and clusters of stations,
- local busses interconnecting stations within a cluster.

6.2 Medium access control

The requirements imposed on the medium access control protocol were:

- to be compatible with a broadcast transmission service,
- to be capable of tolerating the failure of any number of stations,
- to ensure deterministically bounded access times.

Since there was no very strict requirement on performance, the qualitative reasoning set forth in paragraph 5 concerning the relative dependability advantages of competition and consultation protocols led us to choose the competition approach using CSMA techniques. The requirement for a broadcast transmission service rules out CSMA protocols that detect conflicts by timing out on an explicit acknowledgement message and we consequently restricted the choice to collision detection (i.e. listen-while-talk).

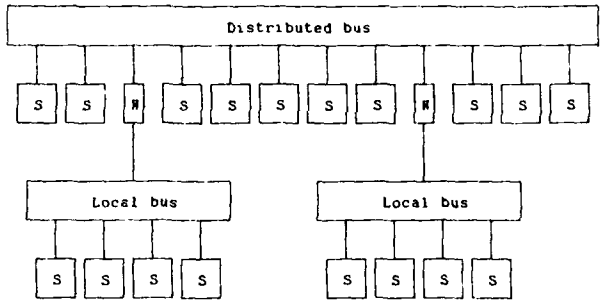


Figure 6: The RHEA network architecture

The protocol that was chosen is the CSMA/FIFO protocol using forcing headers as described in paragraph 5.1.1. Dimensioned for a maximum of 256 stations (8-bit addresses), the forcing header is made up as shown in figure 7:

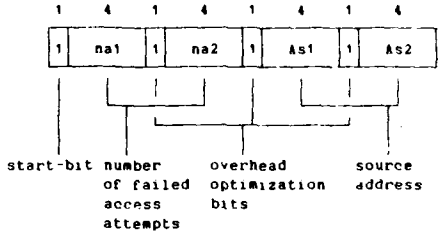


Figure 7: RHEA forcing header

The flow chart of the algorithm carried out by the MAC-machine in each station interface is shown in figure 8. Note that since conflicts are resolved during header transmission, the overall throughput can be increased by using a dual-speed transmission scheme with the information frame sent at high speed. A simulation study of the performance of 12 different channel access algorithms [Powell 81a] has shown that the mean access delay of this technique is of the same order as that obtained with an Ethernet-like CSMA-CD protocol or a token-passing consultation technique.

6.3 Transmission channel topologies

The structures of the transmission channels must be chosen in function of their capability to tolerate multiple localized faults due to physical damage (lesional faults). Their implementations must be either intrinsically secure or use self-checking techniques. Also, in order to facilitate the implementation of broadcast services, the transmission channels must be capable of one-to-all transmission.

In accordance with the specification of tolerance of lesional faults (multiple localized faults), the distributed bus is implemented by a network structure.

The local busses, being intended mainly for interconnection of clusters of geographically localized stations, are designed only for tolerance of independent faults and not lesional faults. In this case, the transmission channels are implemented by star structures.

Figure 9 gives an example of a RHEA topology.

6.3.1 The distributed bus

Three different techniques for network-structured multipoint transmission channels have been investigated [Powell 81b]: a passive, fiber optic network, a signal-switching network and a pulse-switching network.

Each technique is applicable to a particular range of the product: "throughput * network diameter". For transmission over a distance of 100m, at a rate of 100kbit/s to 1Mbit/s, the most suitable technique is pulse-switching.

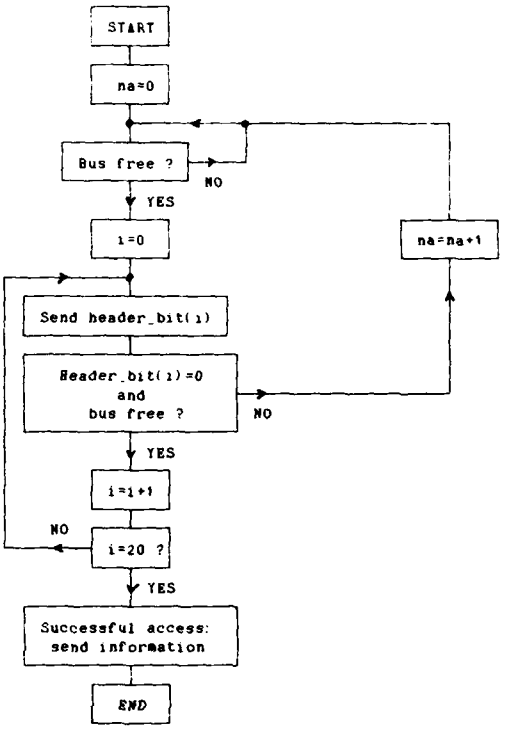


Figure 8: The CSMA/FIFO channel access algorithm
(see figure 7 for content of vector "header_bit")

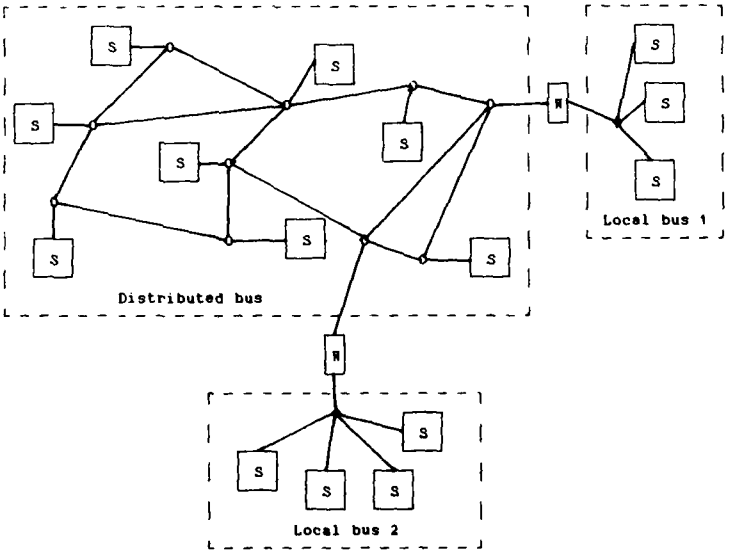


Figure 9: Example RHEA topology

The pulse-switching node represented in figure 10 possesses 4 bidirectional serial links that connect the node either to neighboring nodes or to a station. The input signals from each link are put back into phase by the OR-gate/monostable configuration. Such a technique implies an RZ transmission code and leads to the following maximum transmission rate:

$$D_T = \frac{1}{(4.t_{max}).(1 + L/v.t_{max})}$$

where t_{max} is the maximum propagation delay of a node, v is the signal propagation velocity and L is the maximum link length (this expression comes from the maximum phase difference that must be tolerated, i.e twice the inter-link propagation delay). The collision detection with forcing mechanism can be applied to such a transmission channel if the maximum transmission rate during the conflict detection phase is limited to D_F given by:

$$D_F = \frac{1}{d_{max}.t_{max} + (d_{max}-1).L/v)}$$

where d_{max} is the maximum diameter of the network. Thus, the channel access technique described above can be applied by respecting the constraints: header transmission rate < D_F and data transmission rate < D_T.

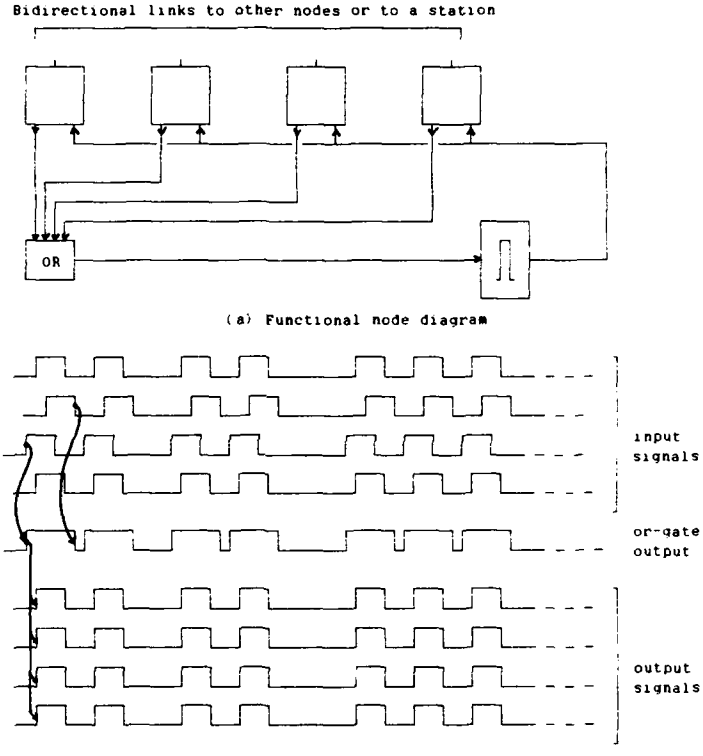


Figure 10: Principle of pulse switching node

The path multiplicity of a network structure enables the distributed channel to tolerate lesional faults due to physical damage. However, it is also necessary for the channel to tolerate faults due to component failures. *The tolerance of such faults in RNEA relies upon the network structure of the channel* in order to implement a distributed self-checking technique with each node controlling its neighboring nodes and disabling them if errors are detected.

Figure 11 gives the block diagram of a node. The functional part of each node and the links between them are duplicated. The pair of signals from each link are compared by a "control block" (figure 12) based on a comparator and a sequential machine. *When the two signals disagree, the control block inhibits* the input and the output on this link. Thus, the control block not only controls the two functional nodes at the other end of the link but also both lines of this link. Faults are tolerated by the redundant nature of the paths in the network. In order to ensure that *the maximum phase difference between the input* signals from different links remains bounded by twice the inter-node propagation delay, it is also necessary to include a pair of "dead line" monitors that inhibit a link if it remains inactive when the other links are active.

The most critical part of any self-checking system is the checker itself. One must be sure that when a fault occurs, the checker will detect it, i.e. there should be no latent faults in the checker. This fault-latency problem is solved here by the design of a control block that, if it fails, will either immediately lead to an error detected by the neighboring nodes or will result in the unaltered propagation of the input signals.

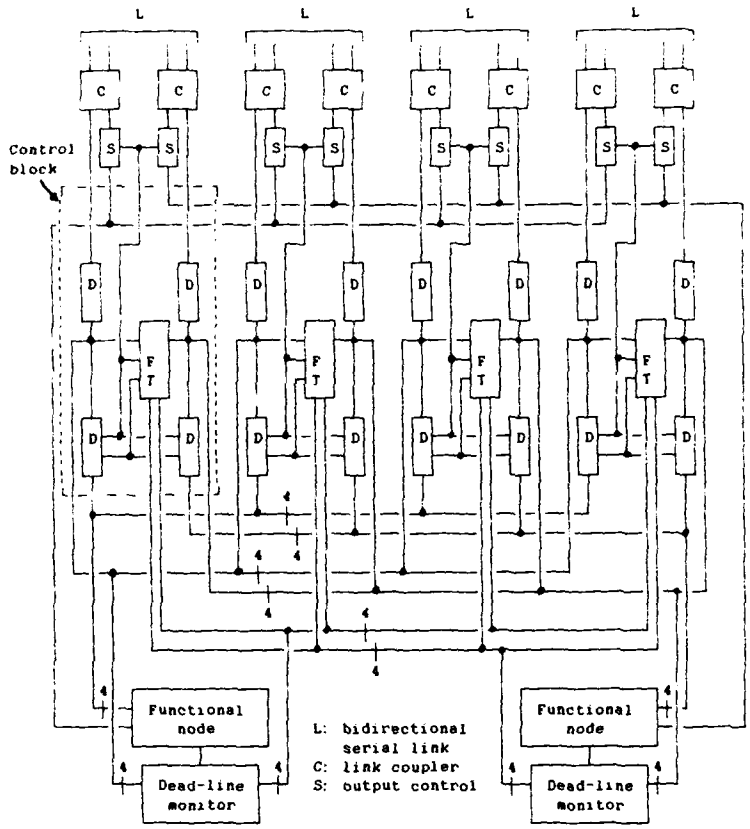
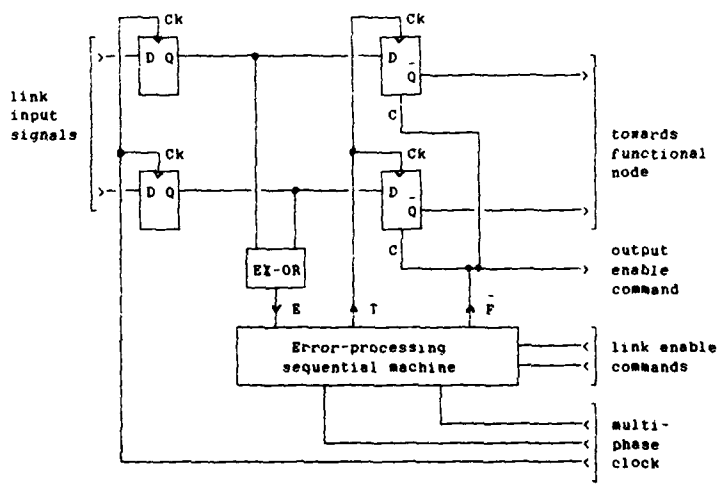


Figure 11: Block diagram of a node



E: error signal
T: transfer control signal
F: permanent error signal

Figure 12: Link control block

Thus, the network nature of the system creates successive barriers to error propagation: if the control block on a link coming from a failed node has also failed then the node containing that control block will also be eliminated from the network by its neighbors. This successive barrier fault inhibition technique is illustrated in figure 13.

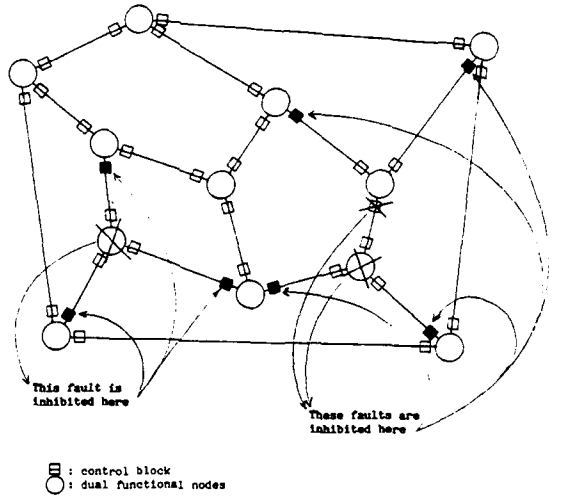


Figure 13: Illustration of the RHEA fault inhibition technique

6.3.2 The local busses

There are two possible implementations of a star-structured multipoint transmission channel used for the local busses: a fiber-optic, n-way coupler (see, for example: [Barnoski 76] or a loosely-coupled, air-cored, n-way pulse transformer.

RHEA uses the latter technique since it is not only possible to realize a central star-node that does not present any hard-core but also, being AC-coupled, such a node prevents propagation of transmitter stuck-at faults. Theoretical and practical analyses at LAAS have shown that a transformer coupling coefficient in the order of 0.01 leads to a transfer function between two windings that is practically independent of the state of the other windings (open or short-circuited) and also furnishes an output signal of sufficient amplitude.

7. ANTINEA: A HIGH SPEED DATA BUS

The avionic LAN summarized in this section was designed in collaboration with Electronique Serge Dassault. The emphasis in this design is on high speed and the resulting system was proposed as a contribution to the work currently being carried out by the High-Speed Data Bus sub-committee (AE-9B) of the Society of Automotive Engineers.

In terms of actual critical values, the objectives that we set out to achieve are:

- Number of stations: ≥ 128
- Transmission distance: ≥ 300 m.
- Number of priority levels: 5 (0-3: aperiodic, 4: periodic)
- Upper bound (deterministic) on the transfer time for priority 3 messages (64 active stations sending 256 bit messages): ≤ 1.2 ms
- Minimum time between periodic messages (16 stations sending periodic messages): ≥ 0.5 ms
- Maximum time for a command/response/confirm sequence: ≤ 1 ms.

As a direct consequence of these objectives, the useful data throughput (excluding protocol and frame overheads) must be in excess of 13 Mbit/s. As a guideline, we are aiming at a system with 20 Mbit/s useful throughput using a raw data rate of about 40 Mbit/s. *It would seem reasonable to offer a specific* command/response mode of access to the transmission resources (if the command, reponse and commit messages *were considered as arbitrary messages, then we would need to achieve a maximum message transfer time less* than 0.33 ms leading to a required useful throughput of nearly 50 Mbit/s).

7.1. Topology

ANTINEA uses a bus architecture in which the physical layer is made up of one or several sub-busses connected to a central repeater. Each sub-bus has unidirectional transmit and receive sections each split into one or several branches; each one may be made up of either co-axial cable or optical fiber. When co-axial cable is used, stations are tapped onto a sub-bus branch using directional couplers. In the case of optical fiber, a tree-configured sub-bus using optical splitters was preferred. A complete system may use a mixture of both technologies (figure 14). The central repeater function is of course a hard-core but its

functionality is so simple that is can easily be made locally redundant. Furthermore, the division into sub-busses (necessary for attenuation reasons) gives an added advantage in that errors due to failure of a sub bus can be confined without bringing down the whole system.

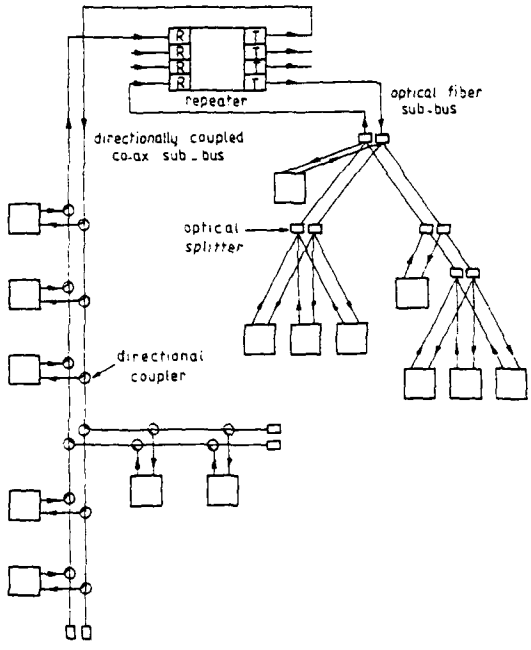


Figure 14: Physical topology of the ANTINEA bus.

7.2. Medium access control

For the reasons stated in section 5, access to the medium is controlled by a token passing protocol that, in normal operation, is decentralized (thus achieving lower overhead than a centralized polling procedure) but for which the virtual ring maintenance functions are centralized and carried out by an active monitor selected from a set of potential monitor or master stations. Apart from the advantages already stated as regards the speed of recovery, the centralized monitor concept can be extended to include special modes of totally-centralized control for purposes of incremental system integration or maintenance.

7.3. Active monitor election

The potential monitors possess a specific monitor identity number MID=1, M. When a potential monitor is powered up, or when it detects that the active monitor has disappeared, it first defers any attempt to transmit for (MID-1) slots. If no other station has started transmitting during this deference interval, it then sends a burst of carrier (preamble) of duration MID slots. If no other transmission is detected after this burst of carrier, then the station assumes the role of active monitor and starts polling in order to initialize the virtual ring. This active monitor election procedure has the interesting property that if all potential monitors initialize the procedure at the same time (figure 15 a) then the time to elect an active monitor is only one slot. If, on the other hand, the start instants are staggered (due to unequal power-up delays) then still only one execution of the procedure is necessary and in the worst case (figure 15 b), an active monitor will be elected within 2M-1 slots of the first initialization instant.

7.4. Priority mechanisms

The priority mechanisms incorporated into ANTINEA are derived from those defined for the IEEE and the ECMA token busses. A total of five priority classes (numbered from 0 to 4) are defined: classes 0 to 3 are intended for aperiodic messages and class 4 for periodic messages.

The periodic message priority class was included in order to allow a restricted subset of active stations to access the medium at well-defined intervals. Restricting the number of stations entitled to this message class means that access delays that can be guaranteed may be much smaller than if the worst-case situation of all stations sending a high priority message had to be taken into account. The synchronous message class is implemented by means of a second virtual ring called the interrupt ring, the normal virtual ring is referred to as the sequential ring (figure 16). Every station possesses an internal timer set equal to the period at which control must be passed to the interrupt ring. If the timer of the station that currently holds the normal, sequential-ring token times out then, when that station has finished sending its current message, it broadcasts an "interrupt mode" message (BIM) that it recognized by the first station on the interrupt ring (e.g. the one with the lowest ID). An interrupt token is then transferred along the interrupt ring until it reaches the last station who then broadcasts a "return from

interrupt" message (RIM) that signals to the station that originally triggered the interrupt sequence to resume operation on the sequential ring. The "interrupt mode" message is also used to reset the internal timers of all stations on the bus (a reliable broadcast need not be assumed).

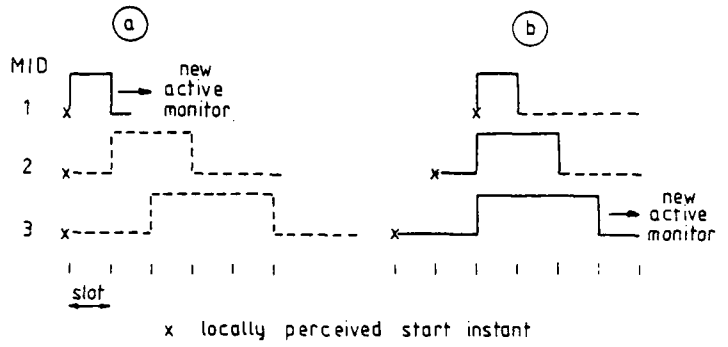


Figure 15: Active monitor election procedure.

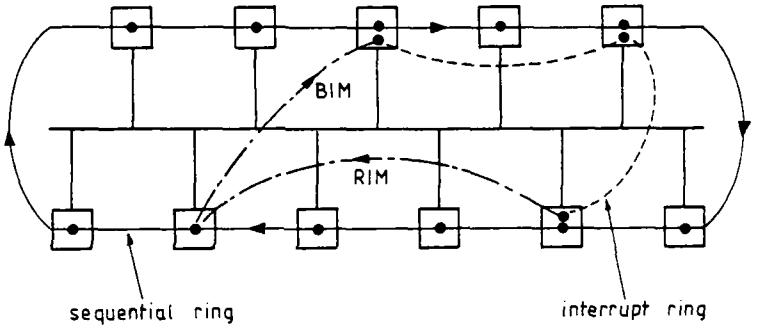


Figure 16: Sequential and interrupt virtual rings.

The aperiodic message priority classes are implemented by a set of "token rotation timers" in each station, denoted TRTi for priority class i, i=0..2 with TRT0 < TRT1 < TRT2. Whenever a station transmits a (sequential) token it resets all its timers. When a station receives the token, it first sends all back-logged messages of the highest aperiodic priority class (class 3). If timer TRT2 has not timed out, the station may then start sending messages of class 2 and so on down to class 0. The maximum time for which a station may send any messages (of all classes) is limited by a further "token holding timer" denoted THT. The interesting property of this implementation of priorities is that if there are no current messages of say, classes 2 and 3, then the total bandwidth of the bus is automatically available to the lower priority classes.

CONCLUSION

It can always be argued that in a particular application, a specially-tailored system will give a better performance than a standard one. However, standard systems should give a reasonable performance over a range of applications (and cost less!). It is only when standard solutions have been pushed beyond their limits that specialized systems or new standards should be envisaged. In this paper, we have tried to outline the essential differences between standard "general-purpose" LANs and those required for harsh real-time applications, and in particular, avionics. The two different and independent projects that have been described were focussed on different aspects of future avionic LAN requirements.

In RHEA, the accent was placed on fault-tolerance and we did not try to achieve very high data transmission speeds. A prototype of the distributed channel with a total of eight pulse-switching nodes was realized and has demonstrated the validity of the distributed fault-tolerance technique. The prototype used Schottky TTL SSI components leading to a node propagation time equal to 250 ns. This, in conjunction with a maximum projected network diameter of 10 nodes and 10-meter links leads to an overall throughput (header + information frame) equal to 350 kbit/s. The throughput limitation is due to the choice of a pulse-switching meshed bus which was in turn due to the need for ensuring broadcast transmission. If only point-to-point transmission is required then a signal-switching meshed bus could greatly improve the resulting maximum throughput (Powell 81b).

In ANTIMEA, the aim was to provide a very high-speed system and the approach to fault and damage-tolerance was restricted to the implementation of independent sub-busses with possible passive branches

into damage prone areas. "ANTINEA" represents our contribution to the current SAE High-Speed Data Bus standardization activity. The new standard may not contain all of the features that we have mentioned but it will nevertheless be quite different to previous standard "terrestial" LANs.

ACKNOWLEDGEMENTS

This work was partially financed by the "Direction des Recherches, Etudes et Techniques" (DRET). RHEA was the object of DRET contract no. 78.352 and was carried out in collaboration with the CROUZET company. The work on ANTINEA was carried out in collaboration with Electronique Serge Dassault (ESD) under DRET contract no. 82.34.452.

Special thanks go to Messrs. P. Plouhinec, G. d'Hervilly and J.J. Mayoux of ESD who contributed very largely to the ideas that we have set forth.

REFERENCES

[Ambrus 85] A. Ambrus, K. Werner: "SILK: system for integrated local communication", Proc. Third European Fiber Optic Communications and Local Area Networks Exposition, Montreux, Switzerland, 19-21 June, 1985, pp. 107-112.

[Barnoski 76] *M.K. BARNOSKI*: "*Fundamentals of optical fiber communications*", ISBN 0-12-079150-1, Academic Press inc., 1976.

[Bean 85] B.E.S. Bean: "'Convergence' towards a metropolitan area network", Proc. Third European Fiber Optic Communications and Local Area Networks Exposition, Montreux, Switzerland, 19-21 June, 1985, pp. 113-117.

[d'Hervilly 84] G. d'Hervilly, J.J. Mayoux, R. Plouhinec, D. Powell, J.C. Valadier: "ANTINEA: an integrated digital transmission highway for aggressive environments - overall specification and general design study", LAAS research report no. 3175. Final report of DRET contrat no. 82.34.452, 30 November, 1984 (in French).

[Penney 78] *B.K. Penney, A.A. Baghadi*: "*Survey of computer loop networks*", parts 1 and 2, Computer Communications, vol. 2, nos. 4 and 5, August and October, 1978.

[Powell 81a] *D.F. POWELL*: "*Performance evaluation and comparison of dependable channel access techniques for locally distributed computing systems*", Proceedings of the 2nd. International Conference on Distributed Computing Systems, Paris, France, 8-10 April, 1981.

[Powell 81b] D.R. POWELL: "Dependable local area networks for control and monitoring", State Doctorate thesis, no. 56. Institut National Polytechnique de Toulouse, Toulouse, France, 23 October, 1981 (in French).

[Powell 82] D.R. POWELL: "Dependability evaluation of communication support systems for local area distributed computing", Proceedings of the 12th. International Symposium on Fault-Tolerant Computing (FTCS-12), Santa Monica, California, USA, 22-24 June, 1982.

[Stiffler 78] *J.J. Stiffler*: "*Fault coverage and the point of diminishing returns*", Journal of Design Automation and Fault-Tolerant Computing, vol. 2, no. 4, October 1978, pp. 289-301.

[Valadier 84] J.C. Valadier, *D.R. Powell*: "*On CSMA protocols allowing bounded access times*", Proc. 4th. Int. Conf. on Distributed Computing Systems, San Francisco, CA, USA, 14-18 May 1984, pp. 146-153.

[Wolf 79] J. Wolf, M. Liu, B. Keide, D. Tsay: "*Design of a distributed fault-tolerant loop network*", Proc. 9th. Ann. Int. Symp. on Fault-Tolerant Computing (FTCS-9), Madison, WN., USA, 20-22 June 1979.

[Zafiropulo 74] P. Zafiropulo: "Performance evaluation of reliability improvement techniques for single-loop communication systems", IEEE Trans. on Communications, vol. COM-22, no. 6, June 1974.

Multi-Computer Fault Tolerant Systems Using Ada

Walter L. Heimerdinger
Honeywell, Inc., Systems and Research Center
Minneapolis, Minnesota 55440

I. SUMMARY

Ada will be the language of choice for a number flight critical applications in the future. Ada incorporates a number of constructs to aid in constructing reliable software, including packages and private data types to manage the visibility of data and strong typing to control the values and operations that can be applied to a data object. Ada also provides constructs to assist in the construction of fault tolerant software. These include tasks and the Ada mechanisms for synchronizing and communicating between tasks, for exception handling and for timing. These mechanisms rely to a large degree on an Ada run time kernel, a set of routines to provide Ada features that can only be implemented while the application program is running. Since the run time kernel is essential for many important Ada functions, it is critical to the reliability and fault tolerance of the application it supports.

Since multiple computer architectures are often used for fault tolerant systems, the implementation of Ada software for multiple computers is an important consideration. Two ongoing projects at Honeywell illustrate the range of options for distributing Ada software on multiple computers. The first approach, which creates a separate Ada program for each computer, makes the least demands on the Ada compiler and run time software. The second approach, which treats all software for a multiple computer system as a single Ada program, requires a specialized Ada compiler and special distributed run time support routines, but separates software partitioning from the application programming.

II. INTRODUCTION

Most flight-critical systems meet the criteria used by the U.S. Department of Defense for an "embedded" computer system; the computer system is an integral part of a larger system, and the system must operate in real time. Also, flight-critical systems usually are multiple computer systems, to provide protective redundancy. The Ada programming language is a logical candidate for the implementation of software for these systems, as Ada was designed primarily for embedded computer system applications, and Ada incorporates a number of features to enhance program clarity and to improve error containment. Also, Ada has been mandated as the programming language to be used for future U.S. Department of Defense embedded computer applications.

The Ada language is a result of a concern by the U.S. Department of Defense over the continuing rise in the cost of software (DoD was spending over 3 billion annually for software at that time). Part of the problem was perceived to be the lack of standardization in the programming languages used for what are now called embedded computer applications. In 1975, DoD sponsored the development of a series of language requirement documents, beginning with STRAWMAN and WOODENMAN in 1975, TINMAN in 1976, and IRONMAN in 1977. An initial slection phase in 1977-1978 evaluated four language designs to meet IRONMAN requirements, selected two of the languages for further development, and resulted in a new requirements document, STEELMAN [1].

STEELMAN specified that the language be strongly typed, with enumeration types, user definable data types, and constraints to limit the range, precision, scale, etc. of a variable. Implicit type conversions were prohibited. STEELMAN further required the ability to encapsulate definitions of anything (including the data elements and operations comprising a type), to make it possible to prevent external reference to any declaration within the encapsulation. Also, STEELMAN required a capability to define parallel processes that could be implemented on multicomputers, multiprocessors, or with interleaved execution on a single processor. In addition to these requirements, STEELMAN required that it be possible to assemble operational systems from separately translated units and that generic types be provided for functions, procedures, types, and encapsulations.

While almost all of the requirements requested had been implemented in some programming language, most of the implementations were in research or prototype languages; no existing production language had ever attempted to simultaneously meet all STEELMAN requirements.

One of the two 1978 language designs was selected to satisfy these requirements. The language was named Ada after Augusta Ada Byron, Countess Lovelace, often recognized as the first programmer. The first Ada language reference manual was published in 1980. After extensive review, including a canvass by the American National Standards Institute, the manual was reissued in January, 1983 [2]. This version of the language, sometimes known as "Ada 83", is the version accepted today.

II. USING ADA FOR FAULT TOLERANT SOFTWARE

Software used in critical applications must be reliable--that is, it must consistently perform in accordance with system requirements. Reliable software is best achieved by careful attention to each of the phases of the software development process, including: requirements analysis, design, implementation, verification, and maintenance or enhancement.

The programming language begins to have a major influence in the design phase of software development. Here, the reliability of software for critical systems can be increased by adhering to a number of quality design principles. The most important is to factor system functionality into modules that can be easily understood and maintained. Each module should meet the following quality criteria [3].

Minimum Coupling--each module should interact with other modules only to the extent needed to create the overall desired function

Maximum Cohesion--each module should perform its designated function without undesired side effects (preferably with no side effects)

Maximum Information Hiding--each module should hide the details of its local data structures and of its local decision logic from its clients. This tends to minimize coupling by limiting the amount of unneeded information propagation. It also promotes cohesion by encouraging the use of local logic and decision structures that are used only for the functions performed by the module.

These objectives can be obtained in almost any programming language; however Ada makes the job easier. Two aspects of the Ada language are especially important for reliable software design, encapsulation and strong typing.

Encapsulation in Ada is supported by the Ada package (which groups related declarations and statements into a single programming unit), and private data types. An Ada package may be a collection of data objects; more often, it is one or more subprograms that can be called from outside the package. Ada packages always begin with a package specification which lists all of the information visible outside of the package. The package specification may be followed by a package body that contains additional declarations that are local to a package as well as statements that implement the functions of the package. The package body provides a place to hide the details of local data structures and local decision logic mentioned above, while the package specification provides the client the interface information needed to use the package. Coupling is limited to items in the package specification.

Consider the following simple example:

```
package SAMPLE is                          --specification
  type ITEM is new INTEGER;                --part of the package
  procedure ACTION (NEW_ITEM:ITEM)
end SAMPLE;

package body SAMPLE is                     --the package body
  subtype INDEX is ITEM range 1..100       --internal variables
  INTERNAL:array(INDEX) of INTEGER         --not visible outside
  COUNT:INDEX;                             --of the package
  procedure ACTION (NEW_ITEM:ITEM) is
  begin
    COUNT:=COUNT + 1;                      --the implementation
    INTERNAL (COUNT):=NEW_ITEM;            --of the package logic
  end ACTION;

begin                                      --initialization of
  COUNT:=1                                 --the package's internal
end SAMPLE;                                --variable
```

Private data types restrict the operations on a type defined in a package specification to a few basic operations or to operations specifically defined in the package specification (the set of operations can be further restricted by declaring the type to be limited private). This hides the details of the construction of the type from the package user, thus limiting the coupling between a package and its users.

Strong typing complements the encapsulation features of packages by limiting the values and operations that can be applied to a data object [4]. Every data object in an Ada program must be assigned a type. By allowing only specifically identified values for discrete data types, the compiler can assist in assuring that only intended values are used. Furthermore, the programmer is encouraged to use names that more fully describe the use of the data item; certainly IF COLOR = BLUE is more understandable to a software maintainer than IF COLOR_CODE = 3.

Basic error containment can be provided with the use of subtypes, in which the values allowed for variables are constrained to the smallest range possible. Further protection is provided by limiting the operations allowed on a given type to a specified set of operations.

Error detection facilities must be supplemented with provisions for error containment. Virtually all programming languages allow a programmer to include statements to handle various error conditions in a module. However, in most languages, the user must provide the logic to invoke these statements when an exception is detected.

ADA SUPPORT FOR FAULT TOLERANT SOFTWARE

Fault tolerant software is a special class of reliable software that obtains a high level of system reliability by managing redundant resource so that the system can continue to operate in spite of the failure of one of its units. Fault tolerant systems are designed in recognition of the fact that even correctly designed systems with valid initial data can become contaminated by errors, possible due to transients.

Fault tolerant software, especially flight critical software must meet several criteria not imposed on conventional software; specifically, fault tolerant software must be:

Real time--the software must complete the processing of critical tasks with adequate timing margins to maintain the stability of control loops and to provide alarm/target information within required time limits.

Self-checking--comprehensive checking of allowed values and operations must be provided, along with efficient mechanisms to invoke error handling routines upon the detection of an error. Most real-time fault tolerant applications also will require the ability to time critical activities; in critical applications, independent timing sources are required. Typical designs for extremely critical applications use multiple processor configurations in which processors test one another or compare results for error detection. These systems require software capable of synchronizing and communicating between software on multiple processors.

Self-healing--the software must use redundant hardware and software resources to increase overall system reliability to required levels. Fault tolerant systems include software to detect, contain, and recover from faults or errors. This requires software to communicate between multiple processors, to synchronize tasks on multiple processors and to restart or to initialize new processes on individual processors while the remaining processors in a multiple processor set continue to provide service.

Most of the operations above take place while the application software is running. This leads us to consider an often neglected facet of Ada, the Ada run time environment. Every Ada target computer supplements the software generated by the Ada compiler with an Ada run time kernel, a predefined set of routines that implement Ada features that only can be implemented while the application program is running [5]. These include the initialization and synchronization of Ada tasks, Ada timing services, exception handling, and resource allocation for dynamically created Ada program units or objects. To the extent these facilities are used in existing fault tolerant systems, they are provided by a separate real-time executive, usually written in assembly language, that is invoked by the application software through special procedure calls. Figure 1 contrasts the relationship between an Ada application program, the Ada run time kernel, the executive (if one is used), and the target computer with the corresponding relationship for a more conventional high order language. When no executive other than the Ada run time kernel is used, the kernel must provide the underlying services for fault tolerant operation.

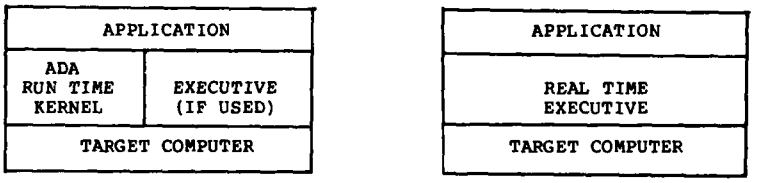


Figure 1. Ada vs Conventional Run Time Support

Fault tolerant systems rely on time for several purposes, including the initiation of processing loops at a repetition rate sufficient to maintain control stability margins, the timing of I/O interactions with sensors and actuators, and the detection of failed units through time outs. Many systems use what is often referred to as a rate structured (or cyclic) executive, which periodically initiates frames of processing activity [6]. To implement cyclic activity in Ada, one would use the Ada delay statement, which suspends the task or subprogram for at least the specified interval. Ada does not guarantee that processing will be resumed at the end of the specified interval, however--the only guarantee is that processing will not be resumed before the end of the interval. If, for

example, a task is suspended by a delay statement and another task of equal or higher priority is enabled, control may be returned to the delayed task long after the expiration of the specified delay interval. As a minimum, the program designer must carefully consider the state and relative priority of all tasks in using Ada timing facilities.

Since time is critical in most fault tolerant applications, it is not unusual for the hardware to provide a fault tolerant clock, based on several independent timing sources. The Ada language provides only one delay statement, however, so the multiple clocks must be managed by the Ada run time kernel. No existing Ada kernel provides such a service.

Ada run time kernels can be expected to check the values of subtypes to detect illegal values during program execution. Errors result in an exception, which, in Ada, interrupts normal program flow. The Ada exception handling mechanism provides a mechanism for fault containment. Exception handling subprograms can be placed in packages where error conditions are likely to arise, or, the exception may be propagated to a more global package that provides coverage for a greater portion of the system. For readability and maintainability, error handling software is best packaged with the module in which the error is likely to be first detected, but the mechanism for invoking the error handler should not cause undue overhead that would jeapordize the ability of the system to meet processing time limits. The efficacy of exception handlers in Ada will depend strongly upon the efficiency of the exception propagation facilities of the Ada run-time support software.

The software may provide error-free service by masking the outputs of faulty processors. Many flight critical systems use replicated processors with voting to provide nearly instantaneous fault masking. Voting, a critial operation in most fault tolerant systems, can be implemented in hardware, as in the FTMP [7] design, or in software, as in the SIFT [8] design.

Hardware voting mechanisms automatically mask out results produced by faulty processors; the software is involved only if recovery or restart operations are performed to restore the faulty unit to service [9]. Hardware voting mechanisms usually require and thus provide a fault tolerant timing mechanism that keeps all processors in lock-step synchronism. Special precautions will be required to initialize the Ada time variable on each of the computers in such a multiple computer system to obtain consistent results.

Software voting is usually accomplished by a "halt-release" mechanism, in which each software subsystem pauses at predetermined points to await results from other subsystems. The software then proceeds if the locally generated result compares favorably with the results received from other redundant units.

The Ada redezvous can be used to synchronize tasks for this purpose. Separate tasks can be provided to read comparison data from each outside source. These can rendezvous with the local task to compare data. A delay statement used as one of the task entries in the rendezvous can be used to raise a time out exception if one of the cooperating tasks fails to communicate within the required time frame.

The above examples illustrate a point that cannot be over-emphasized; because of the number of language features that require support during program execution, such as task management, exception handling, and type checking, Ada software is much more dependent upon the software facilities that complement the compiler to provide support during execution, than is software written in an older, conventional programming language. When Ada is used, the application software must be verified. Note that "validation" by the U.S. Department of Defense Ada Joint Program Office, which is required for all compilers that use the Ada trademark, will probably not be sufficient to guarantee adequate reliability for fault tolerant applications.

At Honeywell, we have been using an interim Ada compiler to develop real time software for experimental applications. In these Ada projects, which are discussed in greater detail later, the Ada run-time software has been the least reliable component of the system. While the more exotic Ada features such as tasking have caused some problems, the most difficult problems to circumvent have been memory management problems, especially stack management problems. For example, if a virtual memory mechanism is not available, the stack space for an individual Ada task is usually obtained from the heap, which is a collection of data areas available at run time for general use. Often, the ultimate size of a task stack cannot be predicted in advance. In this situation, stack overflow can only be prevented by continuous monitoring of stack size, which can require a substantial processing overhead. The run time kernel we use fails to do this, resulting in occasional stack overflows that crash the system. We would not expect these problems to occur in a mature Ada run time kernel, but it is interesting that the interim Ada compiler has been less of an obstacle to software development than the Ada run time kernel.

Ada provides for dynamically created objects, tasks, and subprograms, and a legal Ada compiler and its run time kernal must support all of these constructs. Nevertheless, for the immediate future, the prudent programmer will produce substantially more reliable software by minimizing or eliminating the use of dynamically sized objects and complex tasking relationships. Generally, if a set of Ada packages or subprograms are instantiated only at the beginning of program execution, and if these packages/subprograms continue to operate on objects of fixed size for the remainder of the program, then all will be well.

III ADA FOR MULTIPLE COMPUTERS

The process of distributing fault tolerant software among multiple computers provides one of the best illustrations of the effect of the Ada language design on the design of systems implemented in Ada as well as the importance of Ada run time software. Two ongoing research projects at Honeywell that use Ada to implement distributed computer software illustrate distinctly different approaches to the partitioning of Ada distributed software.

Distributed Computer Testbed (DCT), the first project, is implementing an experimental distributed computer system, with several distributed computer nodes linked by multiple high speed buses [10],[11]. Figure 2 shows the overall layout of the DCT system. In this project, each node executes exactly one Ada program; application software and system software, including the software for interprocessor communications are all implemented as subprograms or library units used by the single main Ada program executing on each node. The application software is written and compiled separately. It uses the Ada "with" clause to obtain the services of the interprocessor communications software, which is implemented as a set of library units. Demonstration software to implement a message exchange system that can continue to operate even after nodes are disabled is also implemented as part of the set of single Ada programs on each node.

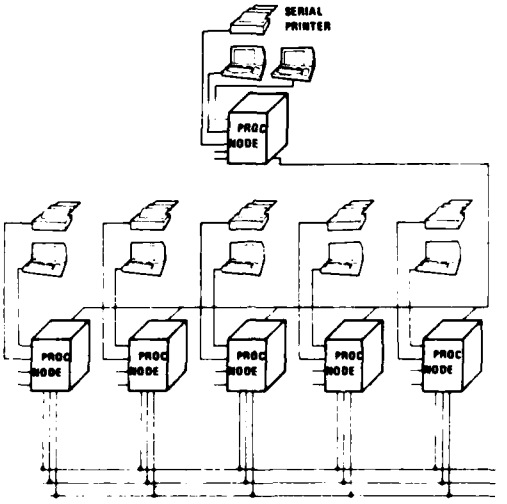


Figure 2. Overall Layout of the Honeywell Distributed Computer Testbed

While this approach makes the least demands upon an Ada compiler and its companion run-time software, it forgoes some of the important Ada facilities discussed earlier. Messages destined for another node are treated as if they were an I/O transaction. Typed objects are subjected to an unchecked type conversion before being transmitted to another node as a chain of bytes, which, upon receipt, is converted back to the appropriate type in the receiving node (again using an unchecked type conversion). No attempt is made to insure that the data type at the sending end matches the data type at the receiving end.

Future plans call for embedding type information in the inter-node message to allow type verification at both ends. In practice, this should cause no problem, especially if the software at each node uses copies of the same library units and the internal representation of a given type is the same on each node. However, the Ada language deliberately does not guarantee uniform internal representations on different machines. For example, it is permissible for an optimizing Ada compiler to use a different internal representation for two instances of the same type if they are compiled in separate Ada programs. This potential conflict must be carefully accounted for in the software verification process.

The communications subsystem is layered; upper layers are implemented in Ada, lower layers are implemented by specialized microprogrammed controllers that run in parallel with the Ada software. The Ada software and the controllers exchange information through queues in a common memory. The communications controllers are high performance devices that can dump a substantial amount of information into the incoming message queues in a short period of time. Consequently, the Ada software must service these queues regularly.

The interim Ada environment being used to implement DCT does not permit the use of interrupts as task entries, so a poller task is used to scan imput queues. This task then interacts with the remainder of the Ada software in the node.

Since each node contains a separate Ada program, the Ada tasking facility does not cover interactions between tasks on different nodes. However, it is possible to construct what is essentially an "extended rendezvous" by suspending a "calling" task when it sends a message to an "accepting task on another node, by suspending the "accepting" task pending receipt of the message, and then by releasing both tasks when the communications system successfully delivers the message. In such a case, the communications system must be carefully designed to generate the appropriate Ada tasking exceptions if a message or its acknowledgement gets lost of garbled. If this is not done, carefully validated Ada software for multi-node fault detection and recovery could fail because of a single communications system fault.

The second project at Honeywell involving Ada distributed software is called Distributed Ada (DA) [12], [13], [14]. This project treats all of the software in a set of distributed processors as a single Ada program. This allows the programmer to use all of the Ada type checking and tasking features in the application software. Interactions between tasks or subprograms on different nodes are automatically managed in conformance with the Ada language by distributed Ada run-time software. Thus, when a task on one node performs a rendezvous with a task on another node, the run-time software operates as described in the earlier "extended rendezvous" example. Because all software in a given configuration is compiled as one Ada program, the problem of differing internal representations discussed earlier does not exist.

The DA approach requires a vehicle to specify the partitioning of the application and the distribution of the partitioned software onto the multiple computers, while treating the application as a single Ada program. DA accomplishes this with an auxiliary language called the Ada Program Partitioning Language (APPL). Figure 3 shows the principal DA componenets and their relationship. The "drts" component is distributed Ada run time support library which provides routines to link the various distributed software elements while enforcing all Ada interface conventions.

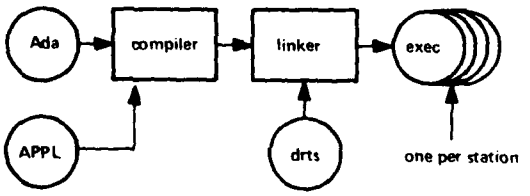


Figure 3. Principal Distributed Ada Components

The ultimate goal of this approach is to produce an Ada compiler with companion distributed run-time software that will allow any Ada resource to be located in a separate node. The requirements that must be met by the run-time software for such a system are formidable; all legal Ada interactions between remotely located resources must be supported using communications links that may not be completely reliable. Of course, the problem of communicating over potentially unreliable communications links occurs in any physically distributed system, but in existing approaches, the application programmer explicitly deals with the communications links and is therefore prepared to provide exception handler to deal with errors that occur when these links are used. In DA, however, every attempt is made to make the partitioning process independent of the Ada program, so an exception due to a communications fault may have to propagate through several layers until an appropriate exception handler is encountered. Furthermore, if the system has real-time processing time constraints, the run-time software must be designed to at least raise an exception if an interaction cannot be completed within a designated time period.

When such a system is built, the resulting advantages will be impressive. An application could be designed and initially tested as a single Ada program operating on a single processor. After the program is tested in this form, the software could be partitioned so that it is distributed over several processing nodes. Interactions between Ada resources would remain the same as in the single processor configuration; i.e. type checking and tasking would remain the same. System timing relationships would change depending upon the relative speed of the processing and communications facilities used.
The DA approach can be extended one step further to introduce the concept of a fault tolerant Ada resource. If we can successfully physically separate an Ada resource from the remainder of an Ada program while interacting with that resource in conformance with the Ada language rules, then we can consider creating multiple copies of that resource,

with each copy located on a different processor. With the appropriate updating and voting mechanisms, specialized Ada run-time software could allow the Ada program to use the resource, even if one of the copies of the resource were corrupted or destroyed. If such a system could be validated, it would provide enormous advantages to the fault tolerant system designer. Redundancy could be applied to individual objects without the need to protect a large number of less critical objects.

Such a system is far from realizable today, but it can serve as an objective as the use of Ada in fault tolerant applications matures. In the meantime, a number of more pragmatic approaches can be used to move in this direction.

IV. CONCLUSIONS

Most fault tolerant computer architectures for critical applications use software executing in parallel on multiple computers to detect, contain, and recover from faults. The Ada language supplies synchronization and timing facilities to manage such multiple tasks, although current implementations of the language are limited to tasks executing on a single computer.

Ada includes a number of facilities, such as task synchronization, package initialization, and type checking, that must be supported by run-time software. System designers that use these features should be aware that these facilities require resources and time, and they may give rise to faults. Poorly implemented Ada compilers or Ada run-time software can seriously degrade the reliability of systems that use them. Conversely, an accurate, verified Ada compiler and fault tolerant Ada run time support software can provide the builder of a fault tolerant system with a number of reliable facilities that otherwise would have to be implemented in the application software.

As time passes and Ada technology progresses, and as a large community of users tests the compiler and run-time software under an ever-widening variety of conditions, the set of troublesome constructs will diminish, and higher quality Ada software will emerge. Then, Ada will take its place as a viable language for the implementation of critical software. In the meantime, the user who is accustomed to assembly languages or to small programming languages should be aware that a significant amount of the functionality of his application system may be embedded in Ada run-time software of which he has no detailed knowledge.

A distinguishing feature of many fault tolerant software architectures is the use of software modules that are executed in parallel on multiple computers. Ada includes tasks, which are program modules that could be executed in parallel on multiple machines with the appropriate compiler/run-time support. If these facilities were available, a fault tolerant software system could be written as a single Ada program, partitioned among the multiple computers.Such facilities are not available today, although research efforts are in progress to develop them. In the meantime, multiple computer softare must be written as multiple Ada programs, one per computer.

The concept of distributing Ada tasks among multiple computers can be extended to distributing any named Ada resource, such as a data object or a subprogram, subroutine, or function. Furthermore, the concept of remotely located Ada resources could be extended to introduce the concept of a fault tolerant resource; a resource which is not only remotely located, but which is also replicated. In such a system, many of the fault detection and recovery mechanisms, such as voting, that are new coded as part of the application software, could be moved into the Ada run time software.

REFERENCES

1. U.S. Department of Defense, "Requirements for High Order Computer Programming Languages," "Steelman," June, 1978.

2. U.S. Department of Defense, Reference Manual for the Ada Programming Language, January, 1983.

3. Joshi, R.D., "Software Development for Reliable Software Systems," Journal of Systems and Software, Vol. 3, pp. 107-121, 1983.

4. Buzzard, G.D., and Mudge, T.N., "Object-Based Computing and the Ada Programming Language," IEEE Computer, March, 1985.

5. Kamrad, J.M., "Real Life Consideration of Ada Runtime Organizations," AIAA/IEEE Digital Avionics Systems Confernece, pp. 472-476, December, 1984.

6. Pratt, K.D., and Sherrill, R.L., "Experiences with the Development of a Real-time Multiprocessor Executive in Ada," 1985 IEEE Conference on Aerospace Electronics.

7. Hopkins, A., "FTMP--A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," Proceedings of the IEEE, Vol. 66, pp. 1221-1239, October, 1978.

8. Wensley, J., "SIFT: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control, "Proceedings of the IEEE, Vol. 66, pp. 1240-1255, October, 1978.

9. Rennels, D.A., "Fault-Tolerant Computing--Concepts and Examples," IEEE Transactions on Computers, Vol., C-33, No. 12, December, 1984.

10. Heimerdinger, W., and Bhatt, D., "DCT--A Testbed Approach to Distributed Systems Research," IEEE 1984 International Conference on Data Engineering, pp. 552-559, December, 1984.

11. Silverman, J., "Communications in a Distributed Computer Testbed," Proceedings of the Fourth International Conference on Distributed Computing Systems, May, 1984.

12. Cornhill, D., "A Survivable Distributed Computing System for Embedded Application Programs Written in Ada," Ada Letters, Vol. III, No. 3, November/December 1983, Revised February 1984.

13. Cornhill, D., "Four Approaches to Partitioning Ada Programs for Execution on Distributed Systems," IEEE 1984 Conference on Ada Applications and Environments.

14. Cornhill, D., "Partitioning Ada Program for Execution on Distributed Systems," 1984 IEEE Computer Data Engineering Conference, April 1984.

DESIGN ISSUES IN DATA SYNCHRONOUS SYSTEMS

by
Gregory M. Papadopoulos
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

SUMMARY

Fault tolerant data *synchronous* systems are ones where the outputs of all correctly operating redundant channels are guaranteed to bit-for-bit agree, independent of whether the channels are clock, instruction or frame synchronous. Data synchronous systems offer a form of fault tolerant processing capable of correctly supporting a very general class of programs. In fact, the redundancy becomes relatively transparent to the programmer of applications for data synchronous systems, making them ideally suited for complex, algorithmically intensive programs that would be otherwise impossible to support.

The various aspects of the design of correct data synchronous systems are examined in detail. These include: *Source consistency*, the requirement that all correctly operating channels receive precisely the same inputs, *Event synchronization*, the problem of keeping the time skew between channels within predetermined bounds, as well system initialization. The unsolved problem of *latent faults* is also presented along with the need for self-test heuristics. Sequential fault tolerant and parallel fault tolerant approaches are contrasted for systems requiring protection from multiple faults. Both hardware and software solutions to these problems are given, *emphasizing system performance* and economy.

The paper concludes with the application of these techniques to the design of triplex fail-operational and quadruplex and dual-dual-dual fail-operational-fail-operational systems.

1. INTRODUCTION

Data synchronous systems employ exact bit-for-bit voting of outputs, offering an application-independent way of protecting against faults [1]. In order that an output disagreement is *only* caused by a fault in the voted-out channel the following conditions must hold:

1. *Determinacy.* The channels are deterministic functions of inputs to outputs. That is, given identical initial conditions then the same sequence of inputs will always yield the same sequence of outputs.
2. *Initial Consistency.* The initial conditions are identical among all of the redundant channels, they bit for bit agree.
3. *Input Consistency.* The input streams, for all time, are identical to all of the redundant channels, they bit-for-bit agree.

Determinacy is relatively easy to guarantee with digital processors, although asynchronous events such as processor interrupts can violate this condition if not properly handled. The second condition is the *initialization problem* which consists of two subproblems: power-up or system restart, and retry or channel restart. The third condition is the *input consistency* problem and is the subject of much attention in this paper. The correct implementation of input consistency requires a different, and somewhat more complex and costly, systems architecture than might be expected to support simple majority voting.

For real-time systems, there are additional requirements that deal with producing outputs and the gathering of inputs in a timely fashion. These reduce in one form or another to *event synchronization constraints*. We are interested in *systems* that guarantee one or both of the following event synchronization constraints.

1. *Bounded Skew. This is a minimum guarantee to make a useful real-time system.* The basic requirement is that healthy channels produce redundant outputs and request redundant inputs within some bounded amount of time of each other. The maximum skew that can be expected among healthy channels is a measure of the tightness of synchronization. Knowing this number allows the construction of correct *timeout tests*.
2. *Total Ordering.* This is a guarantee that the sequence of outputs produced and inputs requested will always occur in the same *order* among the different channels. Having such a guarantee can *simplify* many aspects of interprocessor communication but may substantially complicate processor interrupts.

Solutions to the event synchronization problem fall into two broad categories. *Frame synchronous systems* require the program or operating system to generate periodic points in time when channel resynchronization will occur and interrupts can be resolved. These points are naturally identifiable as the system sample rate or some harmonic, but do require explicit programming constructs and are therefore not completely application independent.

Instruction synchronous systems perform automatic resynchronization and interrupt resolution every *n* instructions. Typically, *n* is on the order of ten instruction times or so. These system have the added advantage of not requiring a regular frame rate structure in the algorithms being performed and thus yield superior application independence over frame synchronous systems. Their primary drawbacks are in the form of fairly substantial design restrictions of the processors themselves: a requirement that all channel activity can be measured in instruction times.

An even stricter form of instruction synchronization is the so-called "microframe" or *clock synchronous* systems. In this case processor resynchronization occurs every *n* clock ticks. This imposes the added design constraint that all channel activity can be measured by a single processor clock.

Input consistency and event synchronization only form *necessary* conditions in the construction of correct fault tolerant systems. The system designer must cope with the possibility that certain critical assumptions may not be valid. They are:

1. *Faults are immediately detectable.* The types of systems required in most flight critical applications are ones that provide high levels of instantaneous reliability as opposed to long term availability. Crucial to meeting this goal is the determination of the system fault state at dispatch time, and continuously for systems that assume sequential faults. The problem of *latent faults* has not been adequately solved but must be seriously addressed by the designer.
2. *Faults are independent.* This assumption is pervasive in most designs and failure analysis. The problem with assuming the contrary is that the analytic design problem becomes intractable. However, systematic or *simultaneous* faults may be experienced in the course of actual operation. The design must be robust for fairly broad classes of simultaneous and transient faults and yet not introduce new failure modes in the process.

In the following sections we shall develop a hypothetical triplex redundant system in order to illuminate these various design issues and some of their possible solutions. We will then examine the implications of a quadruplex system with the attendant reconfiguration problems, the alternative approach of dual-dual-dual, and finally multiprocessors.

2. A TRIPLEX DATA EXCHANGE NETWORK

Figure 1 shows the general topology of a triplex data exchange network. There are a total of six fault sets in the design: three processors and three restoring stages. Theoretically, only four fault sets are required as long as we ensure that the processor hosting a sensor value that is being exchanged does *not* participate in any of the restoring functions. In most cases, the six fault set approach will actually simplify the engineering task and permit higher performance through pipelining. This is the methodology of Draper Laboratories' clock synchronous FTP [2].

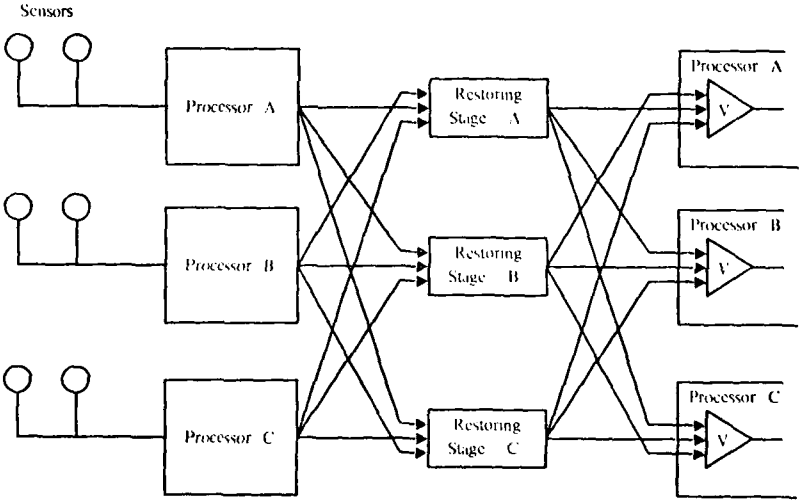


Figure 1. General Topology of a Triplex Data Exchange Network

The system assumes that sensors are attached directly to a processor. In the case of triple-redundant inputs, each processor hosts a single sensor and subsequent data exchanges make all values known to all processors. It should be clear that although the redundant sensor values are associated with each other, they are *not* the same measurement. Thus, a separate data exchange must take place for *each* sensor. In the triplex case, a total of three exchanges. Simplex and dual-redundant sensors might be cross-strapped to two processors under the assumption that the sensor reliability is significantly higher than the processor reliability.

2.1. Systems With Total Ordering and Bounded Skew

The kind of data buffers required between the various elements depends on the kind of event synchronization that the system performs. The easiest case to understand is when the processors produce outputs and request inputs within

a bounded amount of time of each other and the outputs and input requests are totally ordered. Suppose that a sensor value hosted by processor B needs to be read by some program. The data flow of the exchange is shown in Figure 2. The sequence of events are as follows:

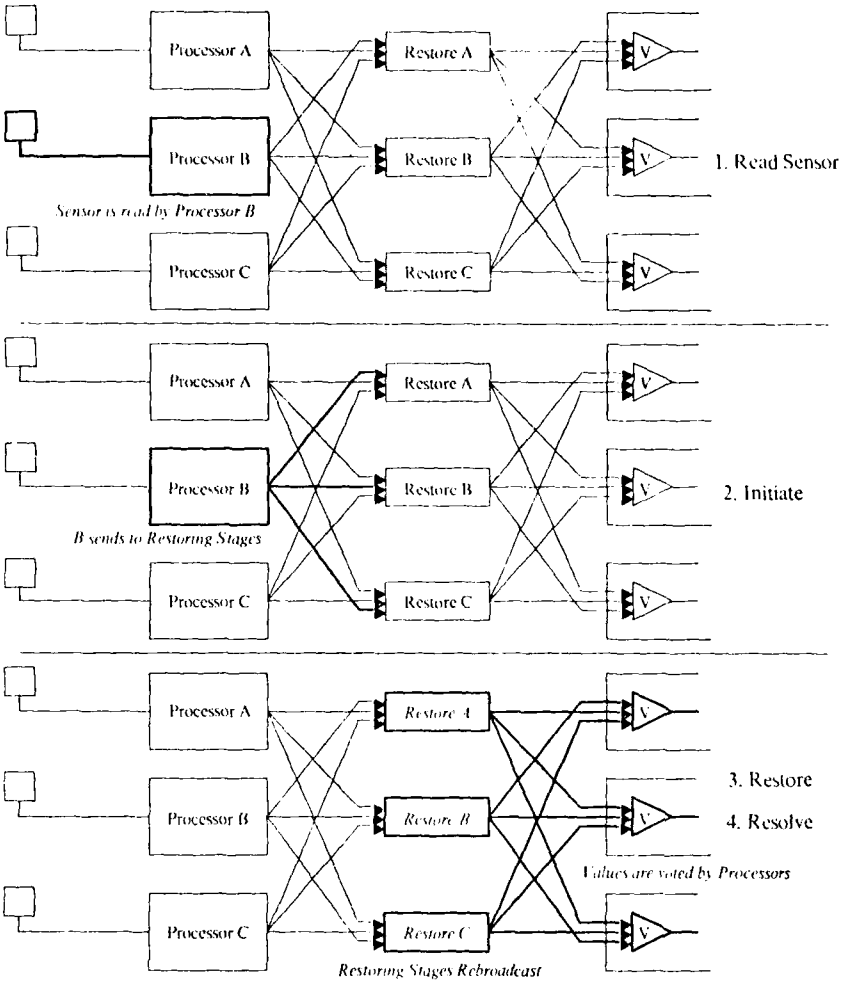


Figure 2. A Sample Data Exchange for a Single Sensor Value

1. *Read.* Processor B reads the sensor whose value is to be exchanged. B must not perform any decisions on the basis of the value it obtained. *It must use the value returned by the data exchange.*
2. *Initiate.* Processor B forwards the value it obtained to all three restoring stages. Similarly, channels A and C send the exchange command "FORWARD B", which instructs the restoring stage that data from processor B should be forwarded. †
3. *Restore.* The restoring stage notices the request to forward B's data value. The data from B is restored (*i.e.*, received and then retransmit) and forwarded back to the processors. Each restoring starts a timer whenever the two FORWARD B commands have *both* arrived. If the timer expires before the value is obtained from B then

† Implicitly, processor B's message also contains a "FORWARD B" command. Implementations might choose to also send data values from processors A and C which are simply discarded by the restoring stages. A and C read "phantom" sensors in the I/O locations where B actually maps its real sensors. This permits all three processors to run identical code that requires no channel-specific modifications. This is the approach of the Draper FTP.

the restoring stage forwards a timeout value instead. The timeout time must be greater than the maximum skew between the processors that is guaranteed by the synchronization algorithm.

4. *Resolve.* The processors bit-for-bit vote among the values received from each restoring stage. If any two values exactly agree then this value is used as the consistent version of the sensor data. If there is a total disagreement then the value is marked as an error value.

At the end of this cycle all healthy processors will contain identical values for the sensor (error or otherwise) in the presence of a single arbitrary fault of either the sensor, a processor, a restoring stage, or the interconnect. An important feature is the placement of the timeout test for the transaction at the restoring stage rather than at the processors. Suppose that B had a skew arbitrarily close to the timeout value (that is, B is faulty). The results of the timeout test in each of the restoring stages must be considered to be a random event. If at least two of the restoring stages timeout then the processors will resolve a timeout for the value. If only one has a timeout then the processors will resolve either a data value, if the input values agreed, or an error token. In any case, all fault free processors will obtain the same result. This would clearly not be the case if the timeout test was performed by the processors.

The buffering needed in the various stages is only one message's worth as long as new values are not exchanged until the previous results are consumed. If this is not the case (*viz.*, burst transfers) then a simple first-in-first-out (FIFO) queue will be required on each of the inputs of the restoring stages and the resolvers (input voters on the processors). The tighter the synchronization as compared to the message generation rate, the shallower the depth required for the FIFOs.

2.2. Systems that Reorder Outputs and Inputs

In the example above outputs and inputs always occur in the same order among all processors. Suppose we were to relax this constraint so that asynchronous events such as processor interrupts can be accommodated. For instance, an interrupt might occur in one processor just before a value is to be exchanged while another processor just initiated an exchange. The interrupting routine will most likely require data consistency exchanges. This will change the order of the exchange requests made by different processors. This scenario is illustrated in Figure 3.

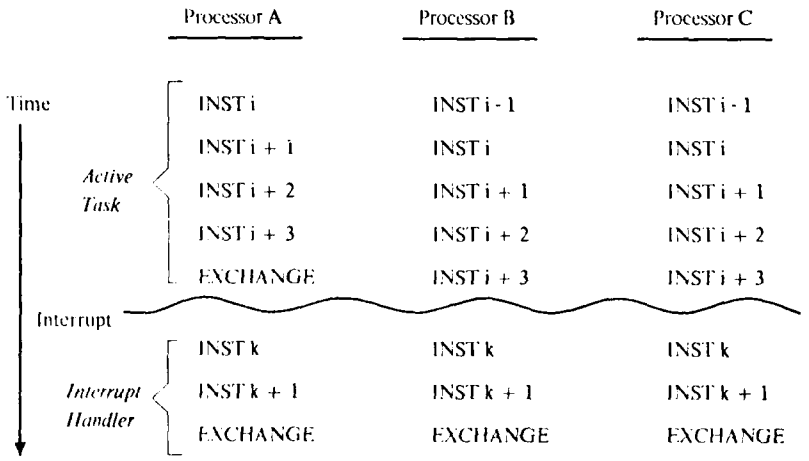


Figure 3. An Asynchronous Event Can Reorder Exchange Requests

The buffers can no longer be simple FIFOs. Instead, each data value to be exchanged must be tagged with a unique identifier. The restoring and resolving stages must match the incoming values on the basis of their tags and then vote among them. The names must not only be unique be also consistent among the processors. One possible solution is to use the taskID of the generating task as the data tag, This assumes that a task will have at most one outstanding exchange. Also, a faulty unit can produce any tag that it wishes. The restoring and resolving functions must be designed such that these bad tags do not induce erroneous matches. This is not difficult to solve but must be considered.

A reasonable implementation, shown in Figure 4, would be to associate a FIFO queue for each task on the inputs to all restoring stages and resolvers. The timeout data must only be accumulated while a task is active, so the restoring stages must be aware of current task. The processors should perform their context switches as close in time to each other as possible. It is also important that attempting to read the results of an exchange is an interruptible process, otherwise

the system may deadlock. Another consideration is a babbling processor or restoring stage that produces more data values than the associate FIFOs can hold. The implementation must take care not to allow an overfull FIFO on one input from corrupting other good input streams. In practical terms this implies that the FIFO management of each data stream be performed independently.

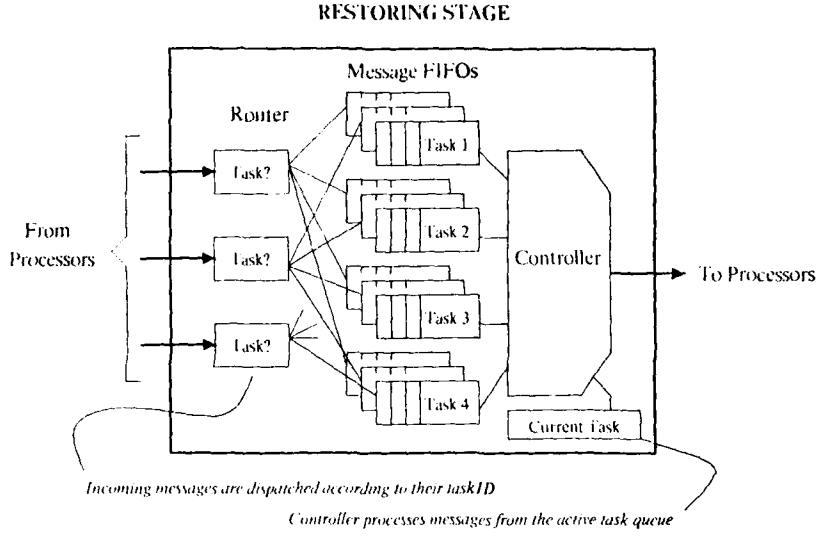


Figure 4. Multiple Message Queues Tolerate Re-ordering

Systems that permit re-ordering of inputs and outputs are more difficult to manage than ones that guarantee a total ordering. The tradeoff, of course, is the effort it takes to provide total ordering versus the effort required to perform consistency maintenance under conditions of re-ordering. Generally speaking, it is currently more profitable to invest in maintaining total ordering rather than provide the required message tagging. As systems become more complex, including multiprocessors and artificial intelligence, this tradeoff will require careful examination—it will become progressively more difficult to constrain applications and distributed operating systems with the total ordering requirement.

3. FRAME, INSTRUCTION, AND CLOCK SYNCHRONIZATION

The efficiency with which systems perform redundancy management is intimately related to the type and degree of event synchronization supported. The *degree of synchronization* is the bound on the worst case skew that can be expected between identical input and output requests among the redundant processors. *Frame*, *instruction*, and *clock* synchronization techniques provide progressively tighter event synchronization bounds. The qualities and implementation considerations of these three approaches will be dealt with in some detail.

Once a designer has adopted a data synchronous design methodology, the approach taken to event synchronization will ultimately have the most far reaching implications for overall system complexity and performance.

3.1. A Synchronizing Exchange

The realization of a synchronizing exchange is quite similar to a data exchange. There are two important distinctions, however. First, the information content of the messages is in their relative times of arrival not, necessarily, in any data they carry. Second, whereas there is only one source of data in a data exchange, it is convenient to exchange synchronizers from all processors simultaneously. As shown in Figure 5, a synchronizing exchange in our hypothetical triplex system takes the following steps:

1. *Initiate.* Each processor broadcasts a synchronization discrete or message to all restoring stages. The content of the message is not important, only that it is a synchronization exchange.
2. *First Selection.* Each restoring stage records the relative time of arrival of the synchronization messages. After the arrival of the second message the restoring stage generates its own synchronization message back to the processors.
3. *Second Selection.* Each processor, in turn, records the relative time of arrival of the synchronization messages from each of the restoring stages, again selecting the mid-value message.

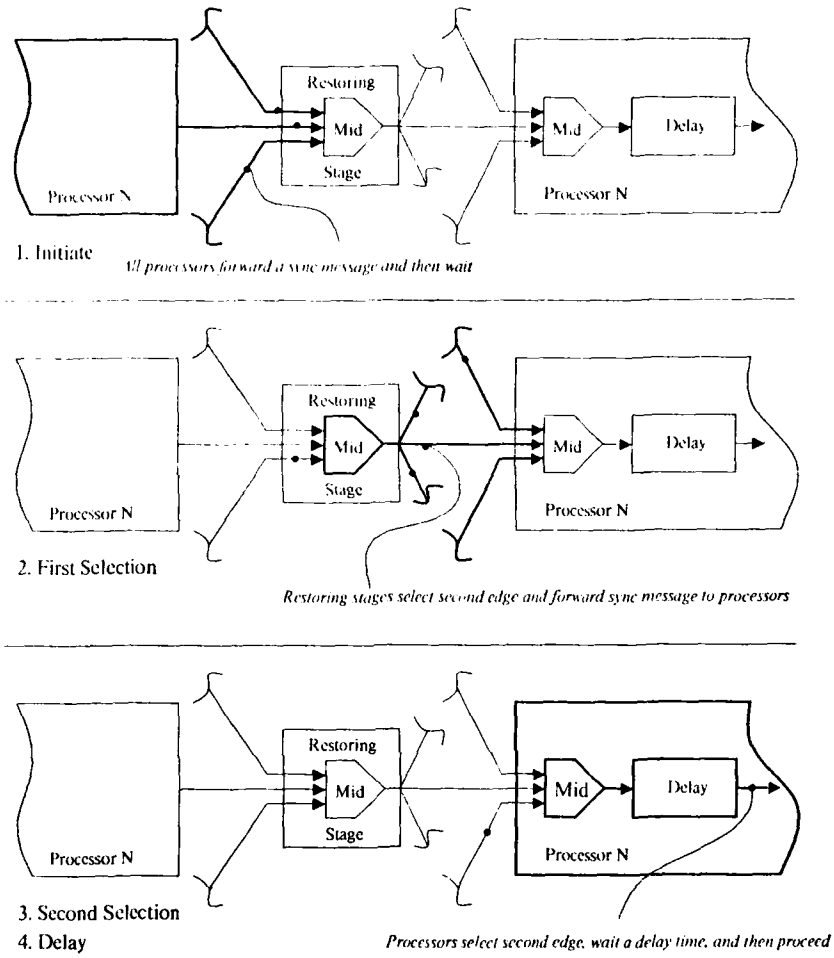


Figure 5. A Sample Synchronization Exchange

4. *Delay.* Each processor then waits a fixed amount of time after the second selection to accommodate a slow processor. This delay should be equal to the worst case skew that could have accumulated since the last synchronisation exchange.

The selection operations are not as simple as might first seem. All synchronisers (which are arbiters) are subject to *metastable states*, conditions where it may take an arbitrarily long time to resolve the second edge. Fortunately, the probability of a metastable state decays exponentially with the duration of the state. It is therefore possible to make these probabilities as low as those of an actual hardware failure.

In practical systems, the restoring stages will apply some sort of filtering to the incoming signals, discarding any input that has a transition outside of a window that the restoring stage expects. This implies that the restoring stage maintain a notion of time as well.

A final problem is that of initialization. The above algorithm only works when the processors are *already* synchronized. It thus accounts for nominal drifts in the processors timebases as well as the failure of any single processor. Initialization can be accomplished in a variety ways, but a common approach is as follows. After power-up or restart each processor attempts a synchronisation exchange at regular intervals. Each processor chooses a different interval, derived somehow from the processor ID *(A, B, or C)*. The different periods must be relatively prime. After a short time, two processors will "collide" in their synchronization attempt. They then choose another period and attempt to synchronise with the last processor.

This initialization algorithm has the problem that it may not work in the presence of a processor failure. It is possible for a failed processor to initially synchronize with the two healthy processors such that these processors are not in synchrony with each other. A more sophisticated algorithm that employs retry (randomly excluding a processor) can be employed if restart is required with a faulty channel.

3.2. Frame Synchronism

The least hardware-intensive approach to event synchronization is *frame synchronization*. In this case the processors perform synchronizing exchanges periodically under software control. This technique seems especially well-suited to control systems where a natural "major frame cycle" can be identified.

It is also common that such systems wait until the frame boundaries to perform data exchanges, usually as block transfers. Figure 6 shows a possible timeline for a frame synchronous system. During Frame i the processors gather input data from their sensors. At the beginning of Frame $i+1$, the processors perform a synchronizing exchange to eliminate any skew that may have developed during the last cycle. The processors then proceed to exchange the data and process it during Frame $i+1$. Finally, the results of the computation are exchanged at the beginning of Frame $i+2$. Note the inherent latency in processing a given sample point.

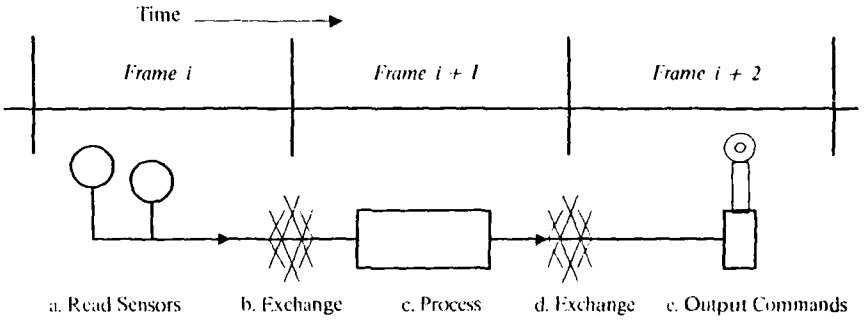


Figure 6. A Typical Frame Synchronous Time Line

A major advantage of frame synchronism is the simplicity of implementation. The synchronization network can be embedded in the data exchange hardware, imposing very little additional design constraints or complexity. The fundamental limitation is the fairly strong assumption about the periodic nature of the application. Certainly all inner loop control systems have an obvious systolic structure. Furthermore most of the computations can be assigned a *rate group*, a sub- or super-harmonic of the major frame rate of the system.

There are several times when the rate structure can break down, making the frame synchronous system vulnerable to inconsistent states among the channels. The common situation is a processor loading condition where the assigned computations can not be completed within the frame schedule. Tasks are then prioritized and the scheduler assures that the most crucial tasks are allowed to complete. This means that the lower priority or background task may be in slightly different states at the end of a frame. For example, a slightly faster processor may have just finished a computation while the others ran out of time. Care must be taken to defer the exchange for the task until the next cycle when all processors have taken it to completion.

But how is the system to know? Suppose there was one slow processors and two fast ones. At the end of the frame cycle they exchange task completion flags. Noting that one processor did not complete, the data exchanges are deferred until the next cycle. What happens if the task still does not complete the next cycle? Do we assume the slow processor failed? Suppose the tasks are of such low priority that they sometimes do not get any cycles during a frame (*e.g.*, a background *self-test*).

It should be obvious that frame synchronous systems depend rather heavily on the behavior of the application being supported. In simple control applications the necessary restrictions are relatively benign. The more complex software systems of the future might find them overbearing.

3.3. Instruction Synchronism

A key observation about two processors started in the same state with identical inputs is that, in the absence of asynchronous events, they will perform the exact same sequence of instructions. We can therefore decide to perform a synchronization exchange after every n instructions and be guaranteed that the very next instruction executed will be identical on all healthy processors. Such mechanisms require direct hardware support and the systems are termed *instruction synchronous*. This is shown schematically in Figure 7.

Instruction synchronism offers considerably more application independence than frame synchronism. It doesn't matter *which* set of n instructions fall between the synchronization points, so the application is not bound to the synchronization period. An important side-effect is that data exchanges can be performed much more efficiently and coded in the instruction stream rather than deferred until a frame synchronization point.

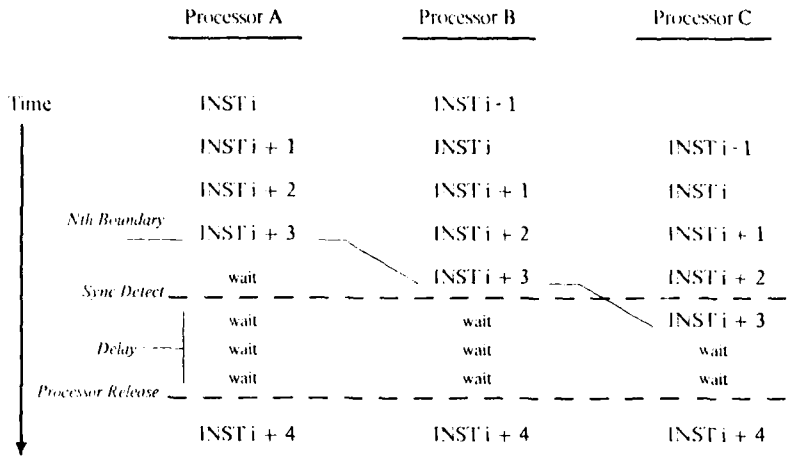


Figure 7. An Instruction Synchronous Control Flow

Instruction synchronous hardware must be able to identify and count instructions as the processor performs them, and more importantly, be able to busy-wait the processor until the completion of the synchronization exchange. This requires that processor activity be regulated in terms of "instruction times." Another difficulty is that complex instruction set microcoded machines can have wildly different execution times that are data dependent. Thus the sequences of n instructions might take markedly different amount of times to perform. The synchronization mechanisms, particularly the timeout delays, must be set according to the worst case (longest) execution sequence. A good number for n would be between 10 and 1000 instructions.

3.4. Clock Synchronism

The strictest form of event synchronization is *clock synchronism*. Clock synchronous processors perform synchronization exchanges every n clock ticks. An approach is shown in Figure 8. A nominal cycle consists on $n-1$ regularly spaced edges plus one "stretched" cycle. A processor that determines itself to be slow omits the clock stretching while a fast processor would stretch the clock any extra cycle. This preserves a constant number of clock edges presented to each processor in each "microframe".

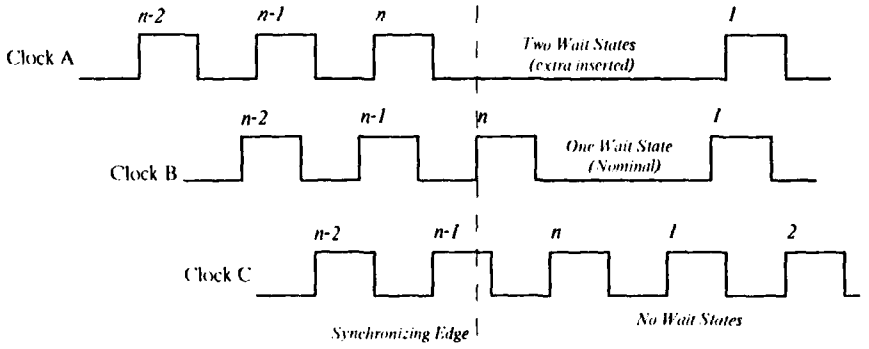


Figure 8. Delaying Edges in Clock Synchronous Systems

There are several advantages of this approach. First, the time interval between exchanges is purely a hardware design parameter and can be selected with considerations of performance and economy, typically ten microseconds or so, completely independent of the application. Second, any event that can be related to the processor clock (*e.g.*, an interval timer interrupt) can be very efficiently processed.

Clock synchronous designs entail substantial restrictions, however. The most serious of which is the requirement that *all processor activity be determined by a single clock.* All memory, bus transactions, and I/O operation must take a predetermined number of cycles. This might pose serious problems for error-correcting memory designs. Typically, error-corrected memory takes longer to complete when correcting an error, an indeterminate event. Dynamic memory refresh

must be scheduled against a global clock. Even simple I/O can be troublesome if the devices introduce wait-states that are dependent on input data—like an analog-to-digital converter.

Overall, clock synchronous systems provide the most transparent and application independent approach to redundancy management, but impose the most severe processor design restrictions.

3.5. Interrupts

Interrupts, the asynchronous change of program flow in response to an external event, need special attention in a redundant system. The problem was developed in Section 2.2—an interrupt might occur at slightly different times in the execution sequence of the processors. This would leave the processors in inconsistent states during the interrupt routine, which generally *look* at various aspects of the system state. Or perhaps an interrupt occurs in the middle of a data or synchronisation exchange, leaving partially completed transactions in the communication network and possibly leading to erroneous timeouts.

There are two distinctly different ways to accommodate interrupts in a redundant system. One approach is to permit the possibility of the interrupt occurring at different execution points but restrict the activity of the interrupt handler such that inconsistencies are avoided. The more general and preferred approach is to guarantee that the interrupt occurs at the same point in all of the redundant execution streams.

The solution by now is familiar—interrupt requests *must* be exchanged in order to be made consistent and then synchronized to the processor instruction streams. Figure 9 illustrates an interrupt exchange network for our hypothetical triplex system. The interrupt discretes are broadcast to a set of interrupt masks. All active interrupt conditions are then ORed and a single interrupt signal is made consistent. This interrupt signal is *only* examined at synchronization points: frame boundaries, after every n instructions or clock ticks, as appropriate. In the case of frame synchronous systems, the interrupt distribution will almost certainly be performed using the data exchange network and only examined at each frame cycle. In this case the signals are more properly termed *interjects*, or controlled interrupts.†

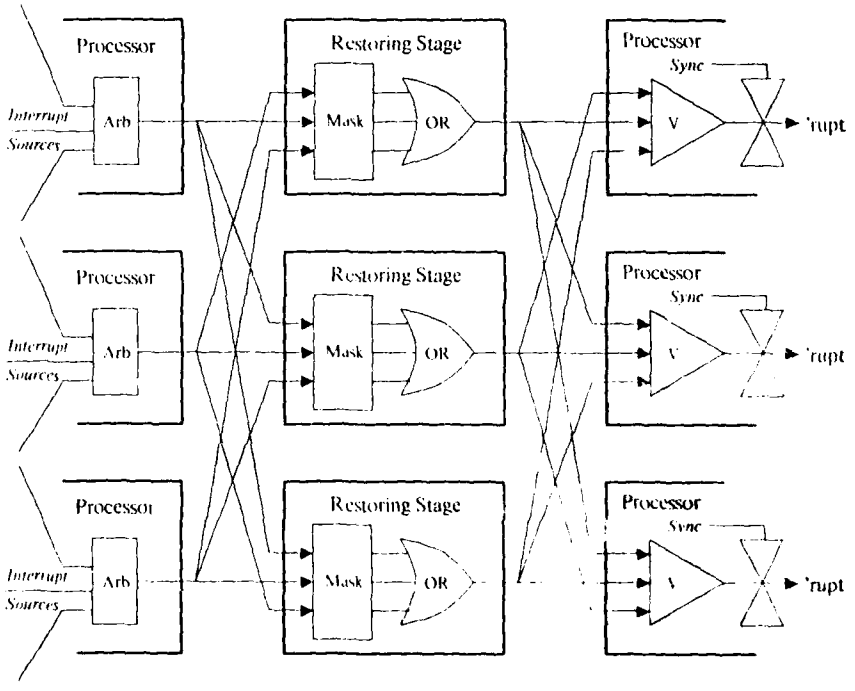


Figure 9. A Sample Interrupt Exchange Network

3.6. Wavefront Synchronism

It seems that the rather simply stated condition of consistent states among redundant channels has become very baroque indeed! For a clock or instruction synchronous machine the following hardware-supported functions are required:

† Of course even a frame synchronous system will usually have some form of hardware interrupt—the interval timers used to schedule tasks and as demarcations for frame boundaries.

A data exchange network, a synchronizer network, an interrupt consistency network. Furthermore, a system that supports direct memory access or distributed memory might require additional hardware mechanisms. These various consistency operations are fundamentaly related, however.

The *wavefront synchronism* concept developed by Hughes [3] is a clever and efficient unification of all of these demands. The basic idea, applicable to instruction and clock synchronous systems, is to encode inter-channel transactions into periodic data exchanges. These transactions would always happen at the synchronization points (every few microseconds), and the "wavefronts" of the redundant messages would be used to judge and effect synchronization adjustments between the channels. The messages would be composed of several fields: a data field for data exchange, an interrupt status field, a DMA request field, *etc.* If during any particular cycle a given field is not used then it is simply set to a null value.

It is possible to encode these messages on say single optical fiber links. Figure 10 illustrates the engineering model of a wavefront synchronizer "box." The wavefront synchronizer for each processor would accept a number of "raw" data on its inputs. Every wavefront cycle the data is sampled, exchanged with other restoring stages, and presented as "good" consistent data on the outputs. A local processor clock is also generated. Thus any data that has the possibility of being inconsistent across channels is simply fed into the boxes' inputs and then read from *its outputs*. The existence of VLSI version a wavefront synchronizer would go far in the realization of general-purpose economical fault tolerant processors.

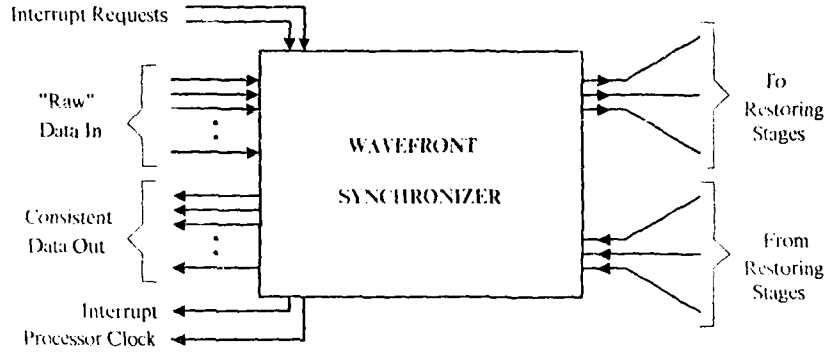


Figure 10. A Wavefront Synchronizer "Black Box"

4. SEQUENTIAL FAULTS

In many flight critical applications, triplex fail-operative redundancy is inadequate. Instead, two independent failures must be tolerated without sacrificing correct operation. If protection is needed against two *simultaneous* failures then five processors and ten restoring stages (two levels of five) would be required. Many times, however, it is adequate to protect against *sequential faults* and thus reduce the number of processors and restoring stages to a more economical level—four of each.

4.1. The Burden of Reconfiguration

One very nice property of our triplex system was a complete lack of reconfiguration logic in the event of a processor failure. The voting logic automatically excludes a single defective processor. Suppose we wanted this system to fail-safe in the event of a second processor failure. Under these conditions we would have to somehow disable the faulty processor to preclude a second processor failure from "colluding" with the first. If the processors were to fail in the same way, then the two faulty processors would vote out the remaining good one.

The same condition holds in the quadruplex case. There is the possibility of *pairwise* agreement, a paradoxical condition where two faulty processors agree but diverge from the two healthy processors. In this case there is no way to discern the good from the bad. Indeed, the probability of pairwise agreement would seem to be quite high. VLSI circuits tend to have design correlated failure modes, for instance metallization opens due to migration. Random failures of VLSI circuits are characteristic of manufacturing defects, not operating malfunctions. Therefore, any system that is intended to tolerate sequential faults must have two attributes.

1. *Timely Identification.* A faulty channel must be recognized within a prescribed amount of time. Of course, the failure diagnosis must be made consistent among the remaining healthy channels. Remember that successful synchronization and data exchanges do *not* imply fault-free channels.
2. *Fault Masking.* After identification, the various consistency exchange mechanisms must be reconfigured to literally ignore a channel that is diagnosed as faulty. *The restoring stages must also be reconfigured.*

The reconfiguration logic substantially adds to the complexity of the exchange mechanisms. Not only must the appropriate configuration registers be provided but there must be some policy for correct initialization on power-up, and for the setting of new values.

The problem is particularly vexing for the restoring stages. How should the relatively primitive restoring stages obtain the configuration data? Should the stages be required to parse data exchanges for a special escape sequence followed

configuration data? Should (yet another) set of dedicated lines be provided to control their configuration? Should the reconfiguration commands be voted from all of the processors or should there be one "host" processor for each restoring stage?

Overall, the reconfiguration problem requires a great amount of engineering effort. It is fair to conclude that quadruplex systems are not instantly more reliable than triplex ones, just because there is an extra processor. *Systems that tolerate sequential faults are substantially more complex, and should not be viewed a simple extensions to single fail-operative machines.*

4.2. Latent Faults

A latent fault is a malfunction that has not yet induced a channel to obtain incorrect outputs. That is, the malfunction has not yet been excited in a way that is observable at the outputs. The problem is that some future input data or program will excite the fault. In a triplex system the consequences are an overstatement of the actual instantaneous reliability of the system. In a quadruplex system two identical latent faults in different channels could cause a pairwise agreement disaster. It is therefore very important that the possible failure modes are routinely excited during operation.

Suppose we desire a quadruplex system with an instantaneous probability of complete failure equal to 10^{-10} per flight hour built out of processors that have a mean time to failure of 10^4 hours. Therefore the probability of any single channel failure is on the order of 10^{-4} per hour. The probability of two successive failures within some small time t is

$$p \approx 12t * 10^{-4} * 10^{-4}.$$

In order that p be less than our target reliability, t must be less than 10^{-3} hours. Therefore, we must not take more than about three seconds to correctly diagnose and reconfigure a faulty channel!

Self-tests are fundamentally heuristic in that it is virtually impossible to generate an adequate fault model against which to design test routines. At a very minimum the self-tests should grade 100% against single stuck-at faults. Not that stuck faults are particularly likely, just that this guarantees exciting each gate in the system.

Unfortunately, with commercially available processors, 100% real-time stuck-at fault coverage is practically impossible, due mainly to the many (effectively) uncontrollable or observable states in the system. Processors capable of such rigorous testing must be designed specifically for the job, most likely employing serial scan path techniques. This is still an open research issue. For the time being, the prudent engineer should employ processors that have a good deal of field experience and avoid the leading edge. In any case, a comprehensive set of test routines must be given top priority. The various aspects of the channel, especially busses, memory, and I/O should have testability features designed-in. For example, all states that can be set must be able to be read. Error detection hardware, such as parity, watchdog timers, *etc.* must be periodically tripped in order to test their correct functioning.

The problems of latent faults and associate self testing are presently unsolved and are extremely challenging. They must be seriously addressed by the designer, especially in systems that are intended to tolerate sequential faults.

4.3. Self-Checking Pairs

Given the attendant difficulties in system reconfiguration, it may actually be cheaper to employ more than the minimum number of processors than to supply extensive reconfiguration hardware. One approach, employed in the Honeywell MMFCS [4], is the *self-checking pair*. The concept, shown in Figure 11, is quite simple. Two tightly coupled processors are run data synchronously. Consistency maintenance with only two processors is almost trivial—a datum is simply braodcast to the other half. The outputs of the processors are bit-for-bit voted. Any disagreement causes both processors to be removed from the system. Importantly, both of the dual-redundant outputs are broadcast to other pairs in the system. A receiving pair votes on the dual-redundant stream, if they agree then the sourcing pairs' correct operation is validated. This removes the output voting logic as a potential single-point failure.

The self-checking pair is viewed as a single self-diagnosing module. The system never "reaches inside" the pair to determine which half actually failed, dramatically simplifying reconfiguration. A fail-operational system is formed using two self-checking pairs, or a total of four processors. This has equivalent reliability properties of a triplex system, however no restoring stages are required.

Similarly a dual fail-operational system can be constructed out of three self-checking pairs, or a total of six processors. In this case there is more than one "listener" of a pair. Thus additional consistency maintenance hardware is required (currently missing in Honeywell's approach). Fortunately, the exchange mechanism is no more complex than that of a triplex system and only needs to make a single bit consistent, monitor compare or miscompare, rather than an entire message.

Self-checking pairs modularize the redundancy management problem and appear to provide an attractive and extensible alternative to the classical triplex and quadruplex approaches.

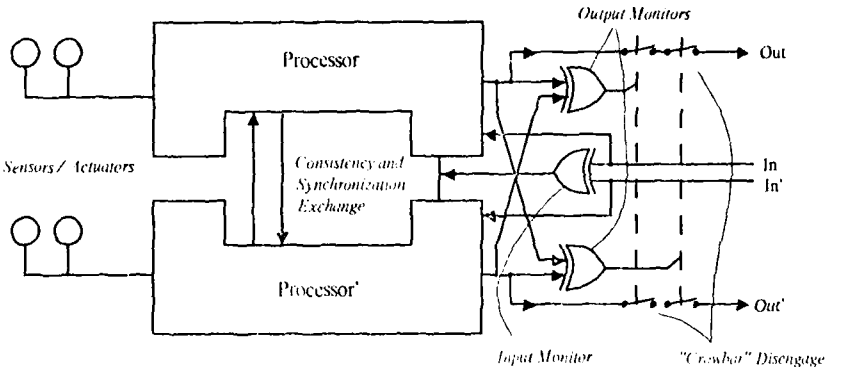


Figure 11. A Self-Checking Processor Pair

5. MULTIPROCESSORS

Avionic *systems* are not made from just a single set of redundant channels, but many cooperating (usually) and diverse components. Suppose that the data *synchronous design methodology is employed in the design of each "box."* How are they to be interconnected in order to preserve the reliability goals of the system?

Several consistency problems, particularly mode changes, appear at the system level as well, but with an important distinction. The condition that makes input consistency difficult is the distribution of a *single* source. In the case of intercommunicating sets of redundant channels, *redundant information sources exist.* A system design should not squander these redundant streams but employ them to simplify global consistency maintenance. At the top level, consistency maintenance is a problem of communication.

The multiprocessor problem doesn't really make sense, however, until the software design issues can be solved. Just as the sampled data formulation led to difficulties at the channel level, it can have a similar nondeterministic effect at the system level. Transactions between boxes should also be viewed as data streams—values are never lost, aliased, or otherwise thrown away. This, however, requires a truly global approach to the software design, an overall operating system. The system must be viewed as running a *single large program* with distributed processes being hosted by the different redundant channels. Sample rates, data representations, and configuration control must be unified. Until this happens, avionic systems will have difficulty exploiting the reliability provided by the different components and truly complex systems, like an expert *pilot, may be impossible to realise.*

6. CONCLUSION

It is imperative that we are able to manage the complexity that new applications, particularly artificial intelligence, are apt to bring to the flight critical realm. In many senses the intuition and techniques of classical flight control systems are inappropriate to these new areas. We will no longer be able to exploit some of the special structures that have lead to the current, albeit very effective, design points. An important goal in managing this new complexity, the aspiration of the data synchronous approach, is to free the applications programmer from the vagaries of redundancy management, and to make hardware fault-tolerance as invisible and robust as possible.

The design of highly reliable digital systems is by no means a solved problem. It is hoped, however, that these notes have provided a perspective that will yield more general-purpose and higher integrity solutions.

REFERENCES

[1] G. M. Papadopoulos, "Redundancy Management of Synchronous and Asynchronous Systems," *Agard Lecture Series No. 143*, October 1985

[2] T. B. Smith, III, "Fault Tolerant Processor Concepts and Operation," *C. S. Draper Laboratory, CSDL-P-1727*, 1983

[3] G. W. Hughes, "The Requirements of a Fault-Tolerant Supercomputer and the Wavefront-Synchronous Dataflow Solution," *Unpublished paper, Massachusetts Institute of Technology*, 1985

[4] K. Driscoll, *et al.* "A Multi-Microprocessor Flight Control System—Phase II Final Report," *Honeywell Systems and Research Center, Minneapolis, MN.*, 1984

This paper was set in Computer Modern Roman 10 point by the author using the TEX typesetting system. The paper was printed on an Epson FX100+ driven by an IBM PC/XT. Line drawings were created with ILLUSTRATE on a Symbolics 3670 Lisp Machine and printed on a Xerox Dover laser printer.

DIGITAL FAULT-TOLERANT FLIGHT ACTUATION SYSTEMS

Howard H. Belmont
Northrop Corporation, Aircraft Division
Hawthorne, California 90250

ABSTRACT

A study was made of the equipments making up a typical flight control actuation system (servo electronics, servo valves, actuators and transducers) to determine where digital technology could replace analog technology for the purpose of providing a more fault-tolerant flight control actuation system.

The investigation involved an analysis of where digital-to-analog conversion should take place between the flight control computer and the analog control surface, and led to an evaluation of several architectural design issues. Among these were how to functionally partition the system, where to locate the servo electronics, the adequacy of military standard serial bus systems for control (versus data) applications, and the feasibility of providing electronics which could survive severe environments.

Several actuation system configurations were evaluated. This led to recommending, as the best development prospect, a locally integrated actuation system consisting of servo electronics, servo valves, actuators, and transducers, interfacing with a digital flight control computer over a serial bus.

1.0 INTRODUCTION

Designs of new, manned fighter aircraft have placed ever increasing demands on flight controls for advanced aircraft performance features. These include interaction and integration with engine controls and with the mission management system for navigational and armament system requirements. At the same time, a revolution in digital electronic computational and communication devices, coupled with evolutions in actuation devices, has created opportunities to improve the performance, availability, and maintainability of the control surface actuation function of flight control systems at lower life-cycle costs.

During the course of the technology survey involved with this study, it was noted that the majority of the flight control technology-based programs being funded were in the areas of digital processing, software, and sensor development. These activities were focused on near-term payoff for production flight control application. A review of technology programs in the actuation area showed that effort was oriented toward the direct drive valve, 8000 psi nonflammable fluids, and electromechanical actuation for the all-electric airplane. Only a limited amount of research and development had been accomplished on digital actuation system technology; therefore, this work and its results are considered to be timely.

The knowledge gap identified was an optimization of the interface between a digital computational flight control computer and the ultimate (analog) control surface.

2.0 TECHNICAL APPROACH

In order to study the knowledge gap identified above, it was necessary to identify the major equipments making up present day actuation systems and define the functions performed by each. These are presented in Figure 1.

Several configurations were synthesized by progressively selecting the digital-to-analog (D/A) conversion point in the equipment between the control processing unit and the control surface, thus increasing the use of digital technology. As these configurations evolved, technical issues arose and these are discussed in the context of each configuration description.

Finally, a configuration was selected to best describe the use of digital technology to create a fault tolerant flight control actuation system.

3.0 CONFIGURATION SYNTHESIS, "BASELINE CONFIGURATION"

The system given as a baseline was a triplex active-on-line system employing per surface, a triple tandem secondary actuator, three servo valves, and three servo processors. The control processors are an analog triplex pair which are self-checked through their servo processor pairs, which also monitor the actuator complex.

The redundancy management, failure detection, and loop closure computations take place in the servo processors.

This configuration is represented by Figure 2. It is significant to note that, for each control processor channel, there are as many servo processors as there are control surface actuators. Thus, for 12 control surfaces, there are 12 channel A servo processors. These 12 channel A servo processors are physically grouped together in the same enclosure with the channel A control processor. The resulting three enclosures (channels A, B and C) may be physically grouped together or dispersed within the avionic bay.

Since the servo processors are packaged with the control processor in the same electronic enclosure and share the same power supply, the interwiring between them can be handled by mother board connections. This is an interconnection density of 10 connections per servo processor or 120 per channel for an aircraft with 12 control surfaces.

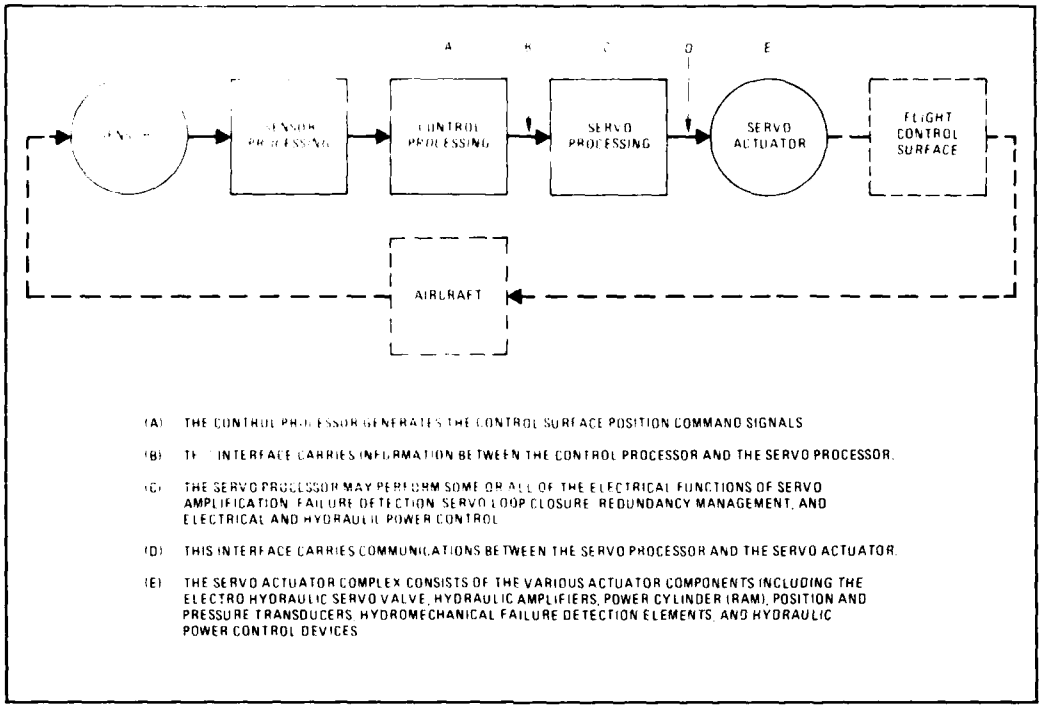


FIGURE 1. ACTUATION SYSTEM ELEMENTS

There are also 48 interconnection wires between each actuator complex and the computer complex. For an aircraft having 12 flight control surfaces, this represents 576 interconnections of aircraft Group A wiring.

3.1 CONFIGURATION I, "DIGITAL CONTROL PROCESSOR"

Changing the control processors to digital from analog allows the use of a digital parallel bus between the control processors and the servo processors within the combined housing. Digital-to-analog (D/A) converters are added to the servo processors as well as a notch filter to smooth out the control processor's sampling rate. The D/A conversion is in the servo processors.

A performance issue, first observable in Configuration I but of concern in all digital processor configurations, is the effect of the waveform emanating from the digital control processor on the electro-hydraulic servo valve. As this discussion will show, these effects were countered by the use of notch filter tuned to the sampling frequency. However, the notch filter introduces a gain change and phase lag which are a strong function of the sampling rate selected.

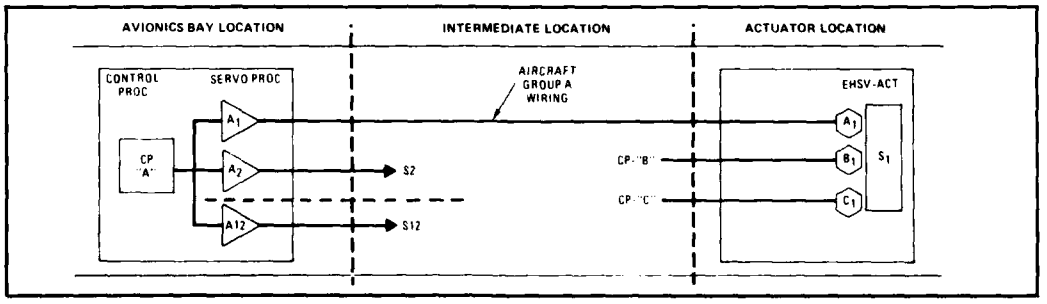


FIGURE 2. BASELINE AND CONFIGURATION I

Processor Iteration Frequency Effects

In a configuration where a digital control processor feeds digital commands to an analog servo processor, the signal from a D/A will be of a staircase waveform, with steps at the command sampling rate of the computer. The rate selected for evaluation was 100 Hz (command update every 0.01 seconds). The effect of this waveform on the actuation device was evaluated using a ramp command.

Figure 3 shows the response of the second stage of the servo valve and the ram to a digitized ramp command (small amplitude steps). These steps are so small that their effect on ram position is insignificant. However, the servo valve spool is almost constantly in motion at this frequency, and such a condition may cause undue wear and fatigue of the servo valve spool and of the hydraulic lines, producing premature failure. In addition, it may create higher hydraulic fluid leakage rates and hydraulic fluid heating.

The transfer function characteristics of the servo amplifier, servo valve, actuator, linear variable differential transducer (LVDT), and demodulator all contribute to the effect. The digital command signal is held constant for one sample period, but the linear feedback signal continues to change. Therefore, the resultant error summation is correct only at the instant of command update, developing a sawtooth waveform. The error waveform, as shown in Figure 3, will pulse the servo valve at the system digital sampling rate.

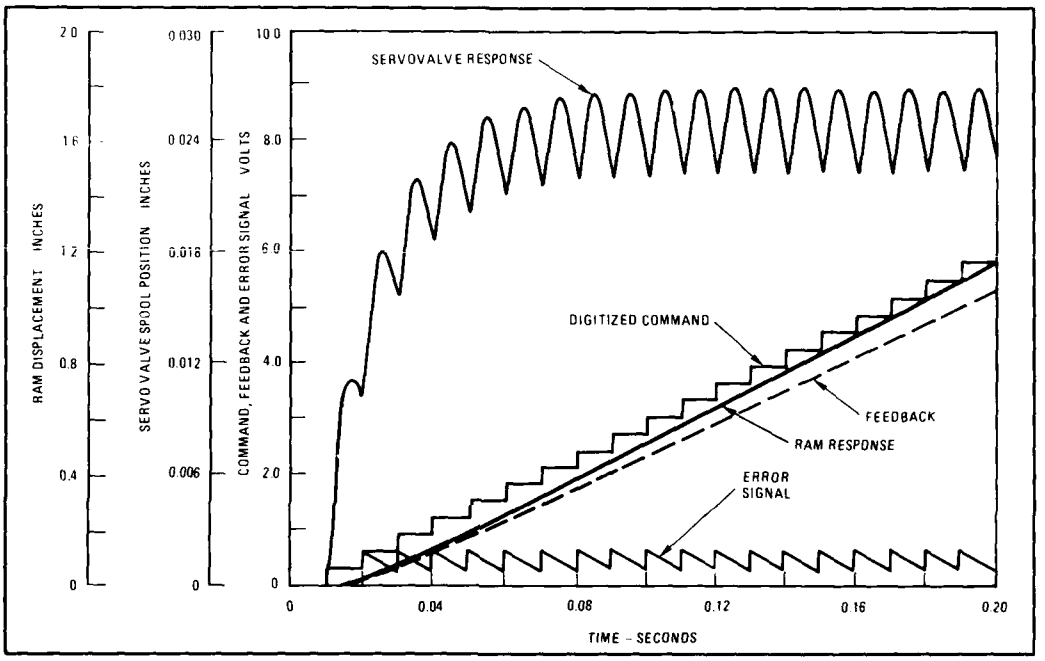


FIGURE 3. EFFECTS OF 100 Hz SAMPLE RATE ON UNFILTERED RAMP INPUT

A notch filter may be used to alleviate this problem. Its transfer function is:

$$\frac{e_{in}}{e_{out}} = \frac{s^2 + \omega^2}{s^2 + 4\omega^2 + \omega^2} \quad \text{where } \omega = 2\pi 100$$

The effect of this filter for a ramp command is shown in Figure 4, which indicates that the filter is effective in reducing the servo valve spool response to the sampling frequency. The ram response is the same as before.

Since serial data bus capacity is directly related to computer sampling frequency, a study was made for two other sampling rates (40 and 10 Hz). Results showed that hydraulic servo valve and actuator reaction was excessive without a filter, and a notch filter solved the problem but created excessive phase lag.

However, another solution is to sum the feedback and command signal in a digital format by converting the analog feedback signal or employing a digital feedback transducer. The digital summation results in an error signal waveform as depicted in Figure 5.

3.2 CONFIGURATION II, "SERIAL DATA BUS"

This configuration, Figure 6, is similar to Configuration I except that the servo processors are located in temperature-controlled intermediate stations between the avionics bay and the actuators. This results in three enclosures each containing 12 servo processors. Each servo processor enclosure has its own power supply which supplies all servo processors within the enclosure.

The control processor-servo processor link is now a serial bus between the control processor and servo processor enclosures and requires a bus controller/transmitter-receiver (BC/TR) at the control processor and a transmitter-receiver (TR) at the servo processor. Parallel bus data distribution is used within servo processor enclosures. Connections from servo processors to actuator (group A – 576 wires) are the same as in Configuration I, although shorter due to the servo processor locations.

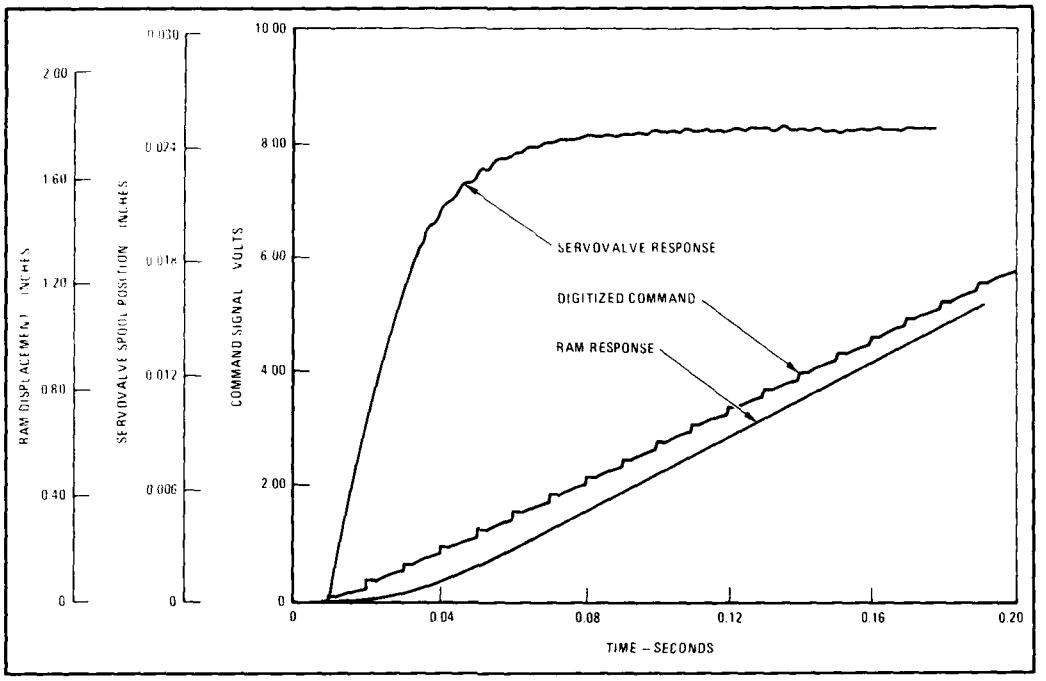


FIGURE 4. EFFECTS OF 100 Hz SAMPLE RATE INPUT WITH NOTCH FILTER

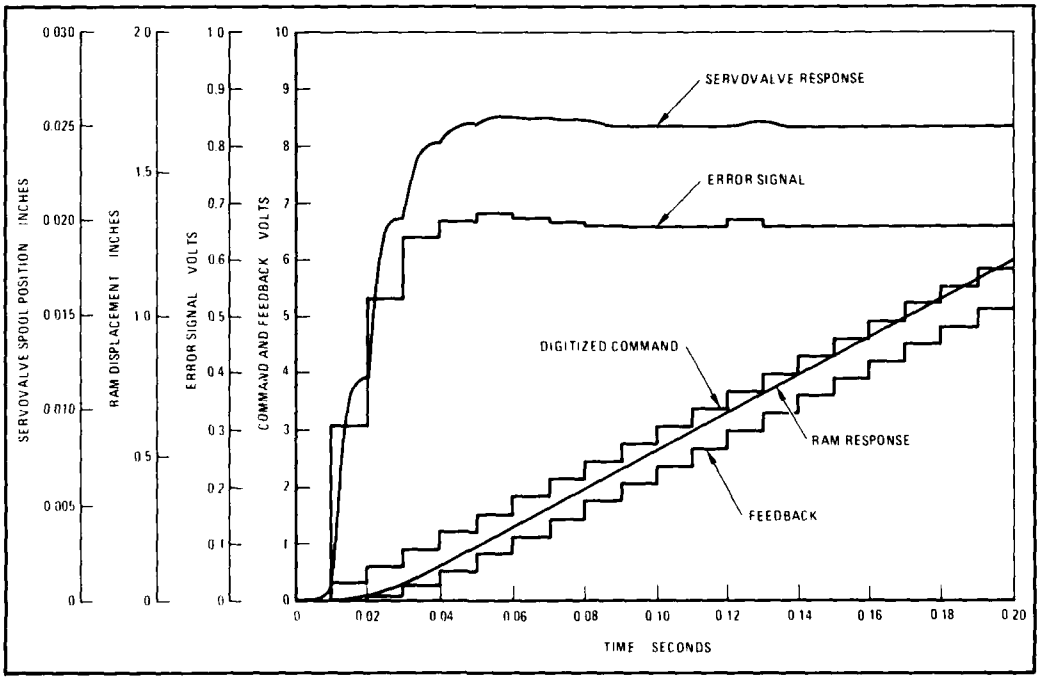


FIGURE 5. EFFECTS OF DIGITAL SUMMATION

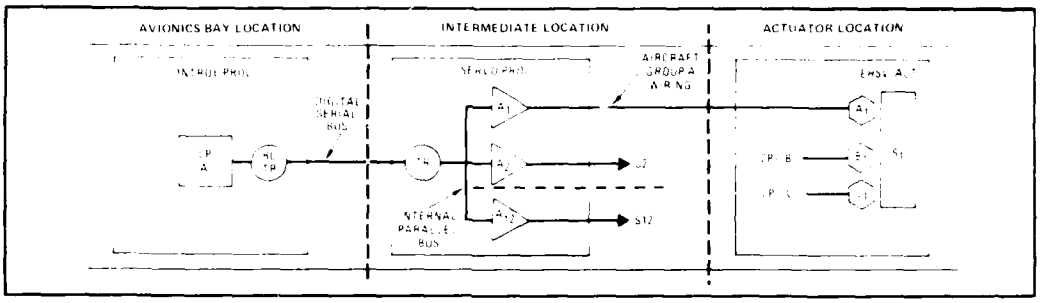


FIGURE 6. CONFIGURATION II

The issue requiring analysis as a result of the relocation of the servo processor is the protocol and percent utilization of the MIL-STD-1553B bus.

Serial Bus Traffic/Functional Partitioning

In the context of Configuration II and those configurations yet to be discussed, the serial data bus requires analysis, particularly as it relates to the adequacy of the MIL-STD-1553B bus. This analysis first describes the ground rules for the application of the MIL-STD-1553B bus. To assess the applicability of the MIL-STD-1553B serial bus to the control processor-serial processor interface, the following assumptions were made:

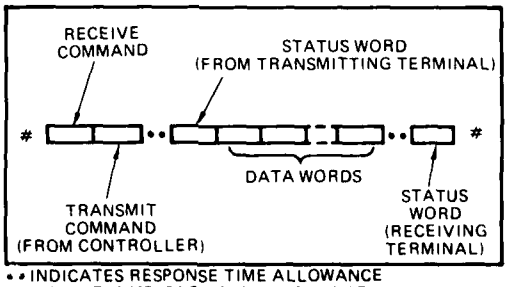
(1) The serial bus is functionally dedicated to the control processor-servo processor interface.

(2) An autonomous bus controller will be utilized that treats the control processor and each servo processor as a remote terminal.

(3) All servo processors require a 100 Hz update frequency.

Data transfer between terminals is directed by a bus controller. Transfer is initiated by two commands which select sending and receiving terminals. The receive and transmit commands include the addresses of the terminal to receive the transmission and that of the terminal which will transmit. The status words (from each) are used to verify the action being performed.

MIL-STD-1553B Format Overhead. The overhead for each transmission, 108 μsec, is calculated based on Figure 7. It shows that the equivalent of 5.4 words (108 ÷ 20) must be sent with each transmission on the bus to define which elements will transmit and receive and to verify the transmission of the data.



•• INDICATES RESPONSE TIME ALLOWANCE
# INDICATES MESSAGE GAP ALLOWANCE

FORMAT OVERHEAD FOR ONE DATA TRANSMISSION	WORDS	TIME μSEC
BUS CONTROLLER (BC) ISSUES RECEIVE COMMAND	1.0	20
BUS CONTROLLER ISSUES TRANSMIT COMMAND	1.0	20
MAXIMUM DELAY		12
TRANSMITTING TERMINAL ISSUES STATUS TO BC	1.0	20
MAXIMUM DELAY AFTER DATA TRANSMISSION (••)	1.0	12
RECEIVE TERMINAL SENDS STATUS TO BC	1.0	20
DELAY BEFORE NEXT TRANSMISSION CYCLE (#)		4
TOTAL NON-DATA TIME IN EACH TRANSMISSION		108

FIGURE 7. REMOTE TERMINAL TO REMOTE TERMINAL TRANSFER

Percent of Bus Capacity Used – We have specified the sampling frequency of the control processor to be 100 Hz (10,000 μ second period). The 1553B format has a maximum allowance of 32 data words per transmission. The number of transmissions possible is then based on the overhead plus the number of data words. We have previously developed the overhead (108 μ sec) for one data transmission (in either direction), and from MIL-STD-1553B we know the length of one word (2 μ sec).

The percent usage of the bus (which should not exceed 70 percent) can be calculated as follows:

$$\text{Percent Bus Utilization} = \frac{\text{overhead time + data word time}}{\text{time allotted}} \times 100$$

$$= \frac{(\#\text{ transmissions} \times 108\,\mu\text{sec}) + (\#\text{ data words} \times 20\,\mu\text{sec})(1)}{10{,}000\,\mu\text{sec}} \times 100 \quad \text{(Eq.1)}$$

Transmissions per sample data period required for Configuration II, assuming that redundancy management and loop closure are performed in the servo processor, are as follows:

Transmissions	Data Words
1 (CP → SP) of:	1 – CP fail-status
	12 – position commands
$\frac{1}{2}$ (SP → CP) of:	2 – SP fail status
	15

Using equation (1) shows:

$$\%\text{ Bus Utilization} = \frac{(2 \times 108) + (15 \times 20)}{10{,}000} \times 100 = 5\%$$

Since only one update cycle is required, the bus has almost 14 times the capacity required (70 percent being the upper limit).

A calculation was made to evaluate the data bus load if a function, such as redundancy management, were performed in the control processor instead of in the servo processor. This would require sending all required data to the control processor. Since a given transmission is limited to 32 words, the redundancy management data words are split up into four separate transmissions.

Transmissions	Data Words
1 (CP → SP) of:	1 – CP fail-status
	12 – position commands
$\frac{4}{5}$ (SP → CP) of:	128 – redundancy mgmt. inputs
	141

Using equation (1) shows:

$$\%\text{ Bus Utilization} = \frac{(5 \times 108) + (141 \times 20)}{10{,}000} \times 100 = 34\%$$

This calculation shows that the data bus would accommodate this data traffic. However, this illustration bears out the benefits of proper functional distribution since the data bus utilization increased from 5% to 34%.

In the discussion of the Digital Fault-Tolerant Actuation System (Section 4.0) it will be seen that the MIL-STD-1553 bus protocol cannot handle the bus traffic except in a broadcast mode. This suggests that this protocol is inadequate for control loop applications where there is a low ratio of data words to overhead.

3.3 CONFIGURATION III, "ACTUATOR INTEGRATED ELECTRONICS"

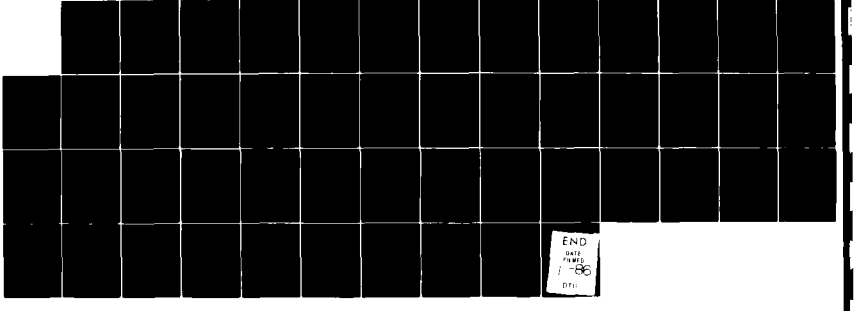
This configuration (Figure 8) is representative of a physically distributed system in which the servo processors are located in the immediate environment of each actuator and are termed "smart actuators." It should also be noted that, contrary to previous configurations where the servo processors were collected in three groups of 12, they are now collected in 12 groups of three.

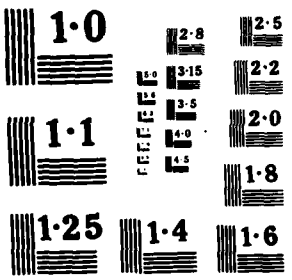
Each servo processor has its own dedicated power supply and monitors. The servo processor to actuator multiwire interface is now internal to the integrated package, or very short if separated physically. The control processor-servo processor serial bus is now distributed to all actuator locations.

This interface differs from Configuration II in that each servo processor must have a dedicated bus interface. The functions of the bus interface have now been slightly reduced. The "word decode and control" function now has only the input for one control processor to control, rather than all servo processors of the same channel.

AD-A181 950 FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION(U) ADVISORY GROUP FOR AEROSPACE RESEARCH AND DEVELOPMENT NEUILLY..

UNCLASSIFIED G L HARTMANN ET AL. SEP 86 AGARD-LS-143 F/G 9/2 NL




1·0
1·1
1·25
2·8
3·15
3·5
4·0
1·4
2·5
2·2
2·0
1·8
1·6

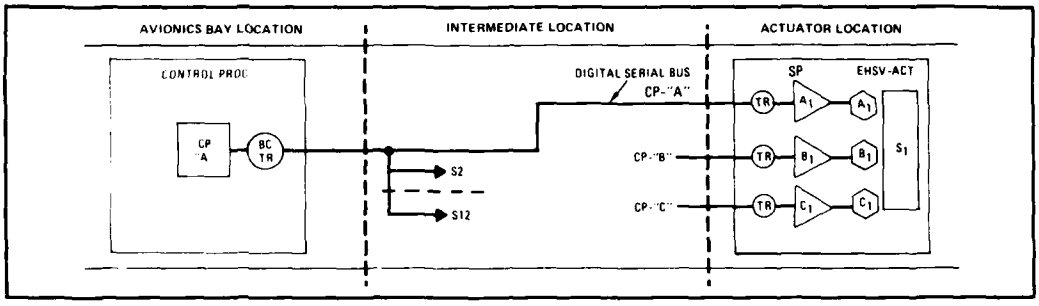


FIGURE 8. CONFIGURATION III

With Configuration III, the economy of common bus transceivers and power supplies has been lost. However, the possibility of some single point failures that can disable one-third of all the actuator channels has also been eliminated. It was noted that a significant difference in weight appeared in Configuration III where, for one channel of three servo processors plus actuator, weight decreased by 25 percent due principally to reduction in aircraft group A wiring.

A technical issue for Configuration III and subsequent ones is that the actuator environment of advanced fighter aircraft will exhibit high temperatures, thus requiring electronics capable of high reliability at these temperatures.

High Temperature Electronics

The Environment. Configuration III has physically distributed the servo processor from the control processor enclosure to integration with the actuator. This poses a problem of either obtaining electronic components which can withstand that temperature environment or creating a controlled environment for the servo processor at the actuator. The latter would require a heat exchange system at each actuator location. From an overall aircraft design viewpoint, this would add weight and complexity as well as an accessibility problem for maintenance. A more attractive alternative would be to develop electronics capable of surviving the uncontrolled environment. The packaging technology for a servo processor which can withstand the vibration, shock, altitude, and contaminants (sand/dust/oil) of such an uncontrolled environment is available today. The remaining environmental problem is temperature.

One source of heat is the valve actuator hydraulic fluid, which is heated as work is performed on it. The valve actuator body itself may experience temperatures from -65°F (-54°C) to +275°F (+135°C) in a type II hydraulic system. Another source of heat comes from aerodynamic heating of stagnant compartment air as a function of aircraft speed. These temperatures have been quoted (for fighter aircraft) as between 10°C to 71°C for subsonic cruise conditions, 132°C maximum for a supersonic 10-minute dash, and 168°C maximum for a 2-minute supersonic dash. Heat generated by the electronics itself is a third source and this can be affected by the circuit design approach employed.

Even under these conditions of temperature it can be shown that reliability can be comparable with present equipments housed in conditioned environments with the development of three technologies. These technologies are: (a) high-temperature electronic component technology, (b) thermal packaging technology, and (c) circuit design technology.

Semiconductor Development Status. Research into semiconductor components involving high temperatures is ongoing at several companies and government organizations. These activities can be grouped into the following categories: (a) those being developed for relatively short-term reliability, as for geothermal and oil drilling applications, and (b) those which are being conducted under government sponsorship for flight critical applications, such as the NAVAIR/NRL/GE program for application to engine controls.

NAVAIR/NRL/General Electric Program. The GE Company's Electronic Laboratory in Syracuse, New York, has been under contract to the Naval Research Laboratory, Code 6810 (Contract N00173-79-C-0010) for the development of high temperature electronics to be used for engine control applications. The program was under the sponsorship of the Naval Air Systems Command. The purpose of the program was to develop a family of electronic components which can operate at 300°C for 10,000 hours, which translates to more than 320,000 hours at 200°C – a probable top specification for flight control.

These components are intended for engine controls mounted on an engine in an uncontrolled, high temperature environment. This presents a similar technical problem as mounting electronics at or on a flight control actuator. The functional operations of the circuitry are also similar to servo driver requirements of the flight control application.

In the first phase of the program, GE Syracuse concluded that Integrated Injection Logic (I^2L) was the best technology available for this application. I^2L is a bipolar, large scale integration circuit technology shown to be capable of operation at 300°C. GE's testing has shown that the major reliability problem is the metallization migration into the semiconductor material at elevated temperatures. A double metallization layer is needed for digital applications at 300°C operation, and the system chosen was platinum silicide/titanium-tungsten/gold. Platinum silicide forms stable ohmic contacts to silicon, and gold interfaces easily with the outside world. A thick layer of titanium-tungsten is needed to separate these materials.

Latter phases of the program called for the design, development, and testing of a family of products, including a microprocessor, read-only memory (ROM), random-access memory (RAM) and possibly a

digital-to-analog converter. GE has predicted availability of components operating at 300°C in the 1985-86 time frame, and almost immediate availability for use at 200°C in prototype quantities.

Electronic Packaging for High Temperature. Some research in packaging technology was done in the 1978-79 period at HR Textron, Inc., using existing components. Hybrid techniques were employed wherein semiconductor chips were attached to the package substrate with eutectics, reducing the thermal path impedance between the microcircuit package and the semiconductor junction from approximately 90°C/watt to 10°C/watt. This work has also been done by the GE company at their Evandale Engine Division in applications related to the FADEC engine control program.

Circuit Design Approach. The use of appropriate switching techniques in a circuit design can reduce the internal power dissipation of the semiconductor devices employed. Acting as saturated switches, these devices operate predominantly in one of two modes: saturation or cutoff. The devices operate for only small periods of time (switching transition) in the active region. It is when operation is in the active region that the most significant junction self-heating occurs. A design approach to maximize the use of switching techniques will generate less junction self-heating than a linear circuit approach.

Driving a conventional electrohydraulic servo valve with switching circuits (e.g., pulse width modulation) is not a new concept. Expanding this design approach to include feedback transducer signal processing and command/feedback summation (loop closure) offers a circuit design approach for servo actuation control with potentially higher thermal environment tolerances than conventional linear concepts.

3.4 CONFIGURATION IV, "DIGITAL ACTUATOR"

This configuration is all digital except for the ram itself. It also employs a microprocessor in the servo processor unit to perform all functions in a digital format. Feedback position and pressure transducers are digital encoders. Digital-to-analog conversion is accomplished in the servo valve or the actuator.

A technical issue, arising with this configuration, is the status of digital servo actuation elements, including the servo valve, the actuator, and the position/pressure feedback devices.

Digital Servo Valves. Three servo valve candidates are selected for discussion. The first is a true digital servo valve. The remaining two are analog valves which are capable of accepting pulse type inputs. Pulse inputs are not true digital (word form) inputs, but are considered here for completeness.

- parallel-binary servo valve
- pulse-width-modulation driven conventional electrohydraulic servo valve (EHSV)
- stepper-motor driven servo valve.

Parallel Binary Servo Valve. The servo valve in this configuration is a parallel wire, binary servo valve (Figure 9) which was designed, built, and tested under United States Air Force contract AF33(657)-8644. This valve had an 8 bit (8 coils) torque motor. While theoretically, performance should match the linear servo valve (given required resolution steps), its complication in manufacturing and the multiplicity of wires to drive it do not recommend it.

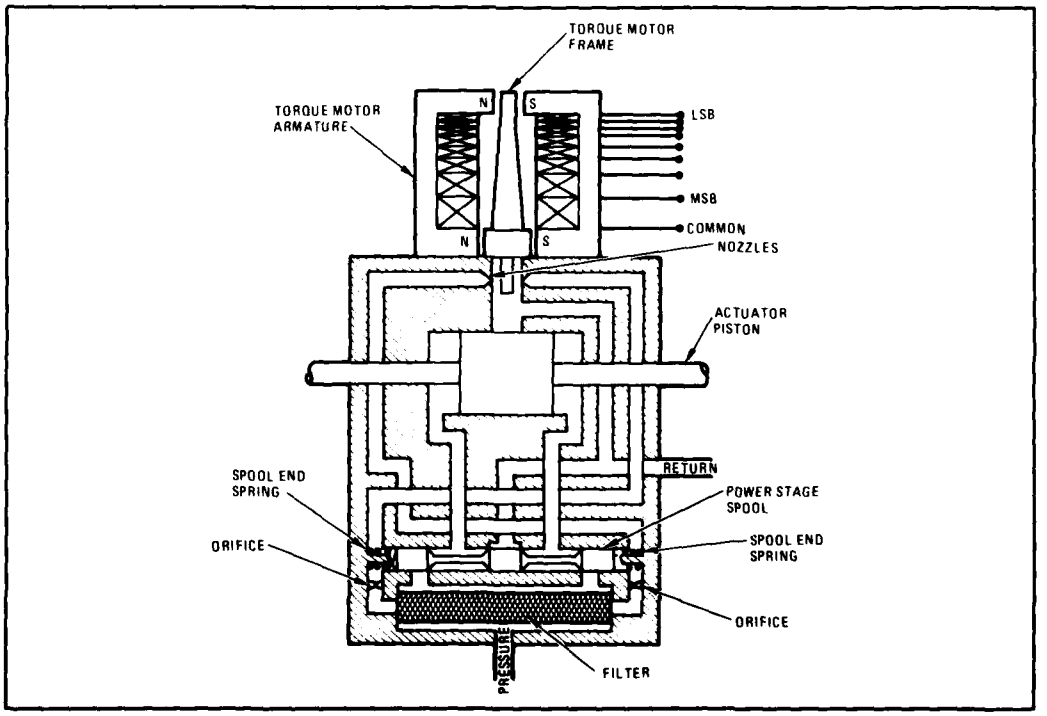


FIGURE 9. PARALLEL BINARY SERVO VALVE WITH ACTUATOR

Pulse Width Modulation Standard Linear Servo Valve. Another approach is to use pulse width modulation (PWM) techniques with a conventional analog electrohydraulic servo valve (EHSV). This concept allows a pulse type of transmission to the EHSV. The driver could be a voltage driver or a current driver. The shortcoming of a voltage driver is that as the EHSV coil impedance changes with temperature, the resultant coil current will change. This appears as an effective forward path gain change in the loop closure path of the servo actuator.

Pulse width modulation fundamental frequencies can be set high enough with respect to EHSV armature natural frequencies to eliminate armature or suspension fatigue. The problem inherent with this approach is limiting the length of the wiring loop between the servo processor and the valve, since the varying current can be a source of electromagnetic interference (EMI). This approach is best suited for a configuration which locates the servo processor at the actuator.

Stepper-Motor Driven Servo Valve. A stepper motor could be used to drive a servo valve spool. Stepper motors accept a continuous control pulse train and increment a predetermined amount of rotation for each control pulse and, therefore, could be applied to an actuation device. It converts pulse signals to exact incremental rotation, and there is no need for any feedback device such as a tachometer (rate) or an encoder (position). If the system is driven open loop, the problems of feedback loop phase shift and resultant instability common to servo drivers are eliminated. Proper application of a stepper motor requires a consideration of:

- Distance to be traveled
- Maximum time allowed for the travel
- Desired static (detent) accuracy
- Desired dynamic accuracy to return to static accuracy (settling time)
- Required step resolution (combination of step size and gearing to the load)
- Total system friction, system inertia, and the speed/torque characteristics of the stepper motor.

A stepper motor accelerates and decelerates with each control pulse even when it is rotating at a maximum speed which causes a velocity change on its output shaft rotation at all operating speeds. Thus a stepper motor would be unsuitable for constant velocity application.

Digital Actuator. If the definition of a digital actuator included techniques in which a digital servo valve was employed but the input signal was an actuator position command (rather than a rate command) then two concepts are possible. In concept 1, the valving element is the digital-to-analog converter. The valving element has a mechanical summing point for the digital command and the actuator position, with the difference, or error signal, determining valve opening (actuator rate). For this concept, the actuator and mechanical feedback are analog, but with the valving element as a summing point the combination satisfies the above digital actuator definition.

In concept 2, the actuator is constructed from a number of series-connected pistons with binary related strokes. For this concept, each piston has its own valving.

Concept 1. Two approaches that satisfy concept 1 will be described briefly. The first approach involves using a servo valve with binary digital or with pulse-width-modulated (PWM) input to the torque motor coils and with mechanical feedback (torque) to the torque motor armature (see Figure 10). The binary digital servo valve could, for example, have multiple coils on a single armature with a binary relationship between the number of turns in the various coils. The PWM approach has considerable advantage and would be a logical choice between the two. Note that the problem of actuator null shift inherent in using the torque motor as a summing point for mechanical feedback has not been avoided by this approach.

This concept has the servo valve as its D/A conversion element and the mechanical loop closure is analog. Since this type incorporates mechanical feedback, a whole new group of considerations different from those in the baseline configuration are introduced. It would, therefore, be very difficult to make valid comparisons. This is not to say that mechanical feedback does not have merit in some applications.

The second approach to concept 1 is one in which a stepper motor produces a stepped input that is summed with a rotation proportional to actuator displacement by means of a mechanical differential. The output of the differential displaces a valve spool that controls flow to the actuator (see Figure 11). In this figure the spline and screw work together to form a differential. This approach has the disadvantage of requiring initialization with some type of position sensor since the output does not have an address. In fact, it may require periodic or continual position updating because of the possibility of lost steps.

Concept 2. Concept 2 consists of a series of actuator pistons connected end to end and having a binary relation between strokes of the different pistons (see Figure 12). The stroke of the shortest piston is equal to the resolution required, and the total output is twice the stroke of the longest piston minus the stroke of the shortest piston. Thus, five pistons stacked in series give resolution of 3.2 percent of total stroke. Separate on-off valving is required for each piston. If back and forth hunting is to be avoided upon the introduction of commands that require extending the long-stroke piston and retracting several of the shorter stroke pistons, the valve flows must be carefully controlled to assure equal traverse time of all pistons under all pressures, at all operating loads, and in both directions. This is not an easy task and could dictate substantial overdesign in load-carrying capability to negate the effect of load. The electrical command is parallel binary, and since digital signals will almost surely be sent to the location of the actuator as serial information (because of the number of wires required for paralleled transmission), actuator mounted electronics to do the conversion would be a necessity.

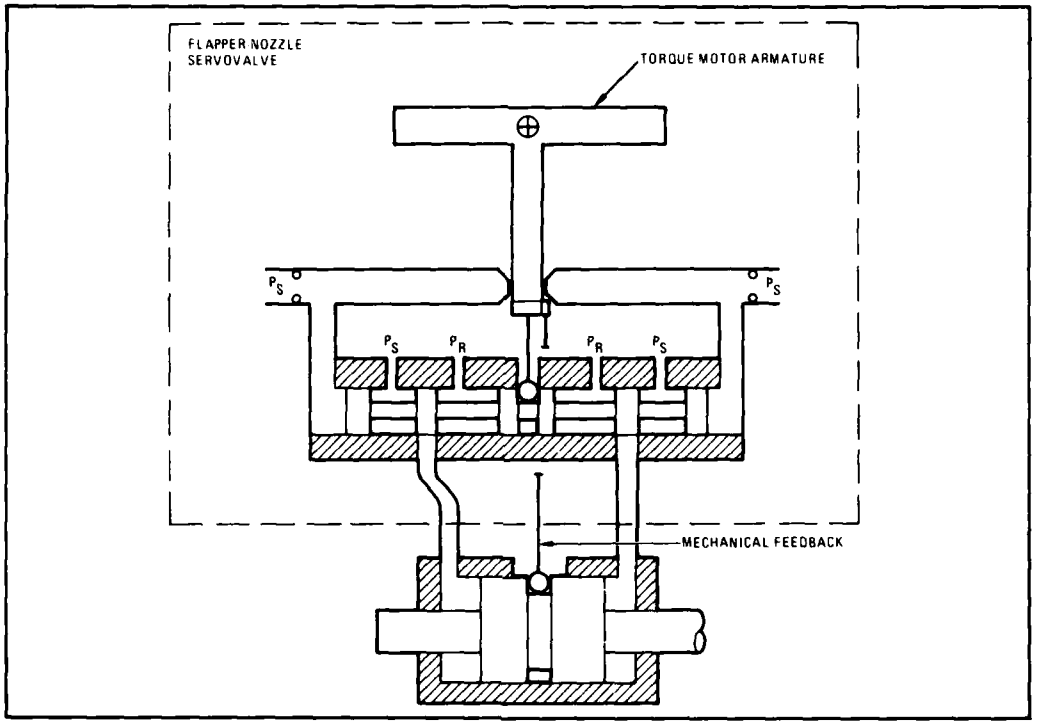


FIGURE 10. BINARY DIGITAL SERVO VALVE – ACTUATOR

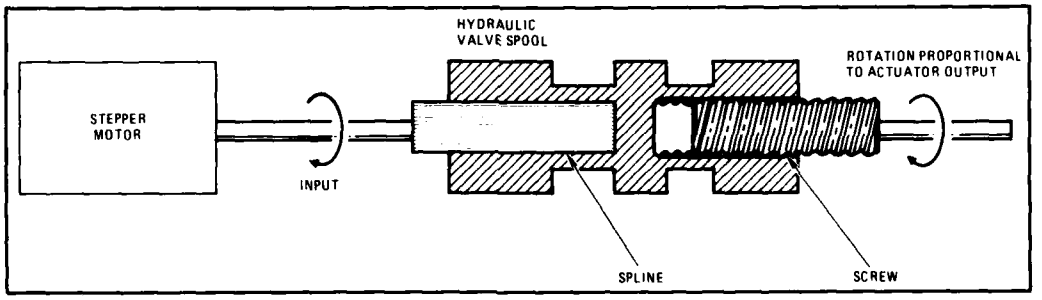


FIGURE 11. STEPPER MOTOR INPUT

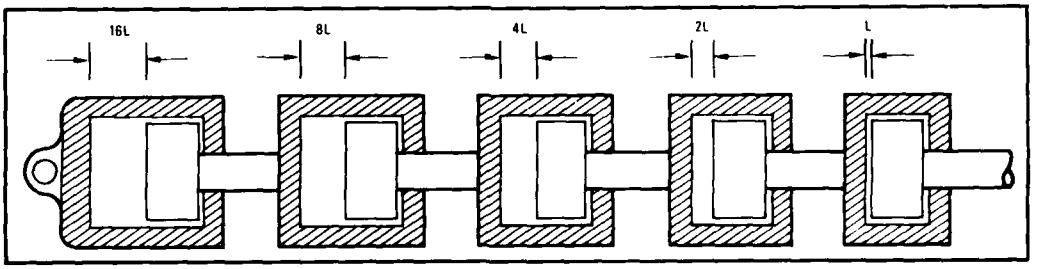


FIGURE 12. DIGITAL ACTUATOR

Note also that there are a large number of sliding seals. This basic type is inherently very heavy and complex mechanically, which alone would remove it from contention for most, if not all, aerospace applications. A positive feature is the elimination of many sources of null drift. This approach exemplifies the difficulties incurred from mechanical digital-to-analog conversion at high power levels.

Digital Transducer. A study of existing "digital" transducers to provide position and pressure feedback has indicated that the mechanisms are based on optics, with pattern-reading principles applied to translate linear motion to digital electronic signals.

The optical/digital transducers rely on their mechanical alignment, but also on the true linearity of the mechanical movement. Some, because the digital scale is not referenced to zero, require initialization. The optical connectors and optical system used require space and alignment which to date has not yielded the same degree of compactness as existing devices.

The present position transducer (LVDT) relies on the accuracy of coil winding and the mechanical precision of the armature to the stator. Its scale factor can be compensated for by means of the electronics which excites and demodulates it, as long as its mechanics are linear throughout. It can be made small for servo valve spool monitoring, or large to monitor long actuator strokes. It can withstand the actuator environment.

Perhaps the biggest deficiency of the present LVDT is in the number of interconnections between it and its excitation and demodulating electronics in the servo processor, and the wire weight required when the servo processor is located in the avionics bay. This objection can be ameliorated when the servo processor is located at the actuator. In addition, digital excitation and demodulation techniques have been developed to effectively "digitize" the existing LVDT devices.

4.0 A DIGITAL FAULT-TOLERANT ACTUATION SYSTEM

From the study of architectural issues, a configuration was selected which incorporates digital technologies with the highest potential for a fault-tolerant system. It is also one which can be presently implemented, given that a temperature environment can be created, or that electronic elements can be provided that can withstand the environment and can be acquired at a reasonable cost.

This configuration demonstrates an approach to creating a voting plane at the servo processor/actuator location. This is accomplished by employing a microprocessor pair in the servo processor units and connecting all serial buses to each servo processor. The actuator elements are standard electro hydraulic servo valves driven by analog servo drivers.

Physical Location of Elements. The three servo processor pairs associated with each actuator are grouped at that actuator location.

Power Supply. Each servo processor has its own dedicated power supply.

Interconnections. Control processors are now linked to all servo processors via three serial buses. Interconnections between servo processors and their actuator complexes are by means of short wire interconnects.

Triplex Channel Mechanization. Figure 13 shows the block diagram as a multi-actuator system. As may be seen, all servo processors have access to the serial buses of all control processors. This creates a system which allows full system performance with any control processor pair and any actuator servo processor. In previous configurations, for an actuator to be operational, the control processor and servo processor in the same channel (A, B or C) had to be operational.

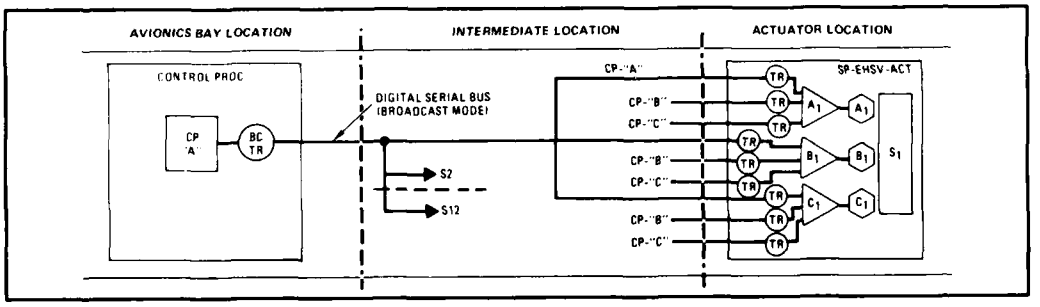


FIGURE 13. DIGITAL FAULT TOLERANT ACTUATION SYSTEM

Since each servo processor now is connected to all control processor-servo processor buses, the failure status of servo processors for channel reconfiguration purposes is now provided via the buses, rather than by means of dedicated wires between servo processor channels. As in previous configurations, the servo processor to actuator link is by means of short wire interconnects.

Serial Bus Interface

Figure 14 shows a block diagram of each servo processor's interface to the serial buses. The receiver from each bus feeds dual-buffered serial-to-parallel converters and dual-word detectors, a set for each internal microprocessor bus. Each word detector drives its associated interrupt controller and each serial-to-parallel converter feeds its associated internal bus through a buffered register. A transmitter for each serial bus is supplied from both internal buses with data. The transmitter control arbitrates access to the transmitter.

Each servo processor now receives the control processor commands and the status for all three actuator channels. The mid value of the three commands is used as the position input command for the actuator servo loop closure. Each servo processor's fail status is transmitted at each update cycle to each control processor. This status transmission is monitored by the other servo processor channels of the actuator to implement reconfiguration in the event of a servo processor failure.

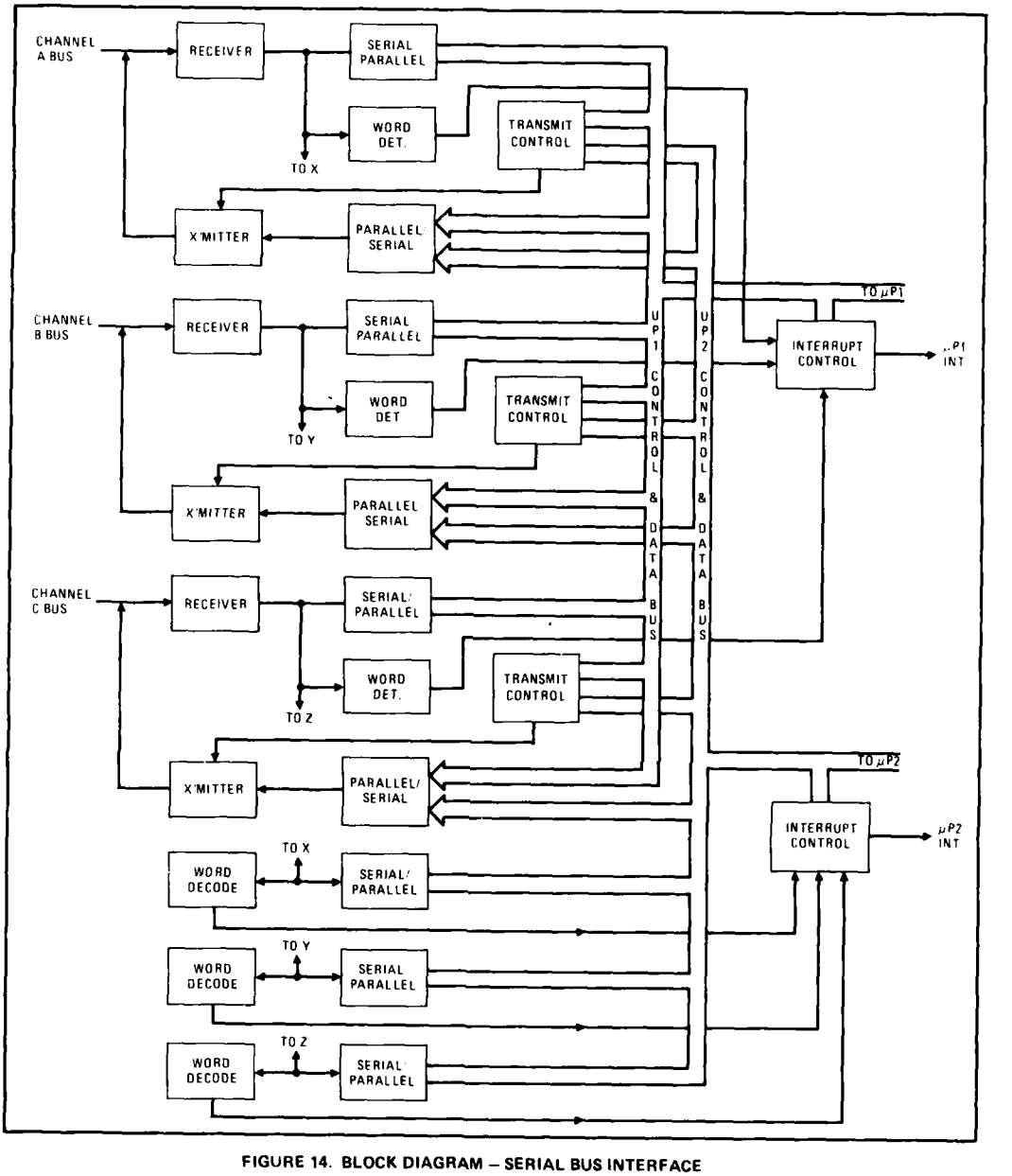


FIGURE 14. BLOCK DIAGRAM – SERIAL BUS INTERFACE

All analog values of the ram, EHSV, and differential pressure (ΔP) LVDTs (in normally on-line servo processor) are multiplexed into one analog-to-digital converter and fed into the μP. The discrete output of the LVDT monitors are combined in a buffer register and also fed into the μP. With this information, and the last command output from the μP known, the EHSV model may be implemented and the results compared to the actually sensed EHSV position in the μP. The μP program then decides the status to be sent to its "driver" register, in light of the (1) model comparison, (2) serial bus fail status, and (3) power supply fail status.

Bus Utilization. Attempts to use the receive-transmit-status word mode of MIL-STD-1553B showed lack of bus capacity, forcing use of the broadcast mode. Using the broadcast mode, transmissions per sample data period for one bus are as follows:

Transmission	Data Words
12 (CP broadcast to each surface's SPs)	1 – reset 1 – position command 1 – CP fail status

Transmission		Data Words
12	(SP"A" broadcast to CP A and to all SPs)	1 - SP"A" fail status
12	(SP"B" broadcast to CP B and to all SPs)	1 - SP"B" fail status
12	(CP"C" broadcast to CP C and to all SPs)	1 - SP"C" fail status
48		(6 x 12) = 72

Using equation (1) page 6

$$\% \text{ Bus Utilization} = \frac{(48 \times 108) + (72 \times 20)}{(10{,}000)} \times 100 = 66\%$$

This shows that the bus has the capacity in a broadcast mode. What is sacrificed is the verification that transmission has been received.

There was no need to consider redundancy management at the control processor for the actuation system in view of the use of microprocessors in the servo processors.

Servo Processor

Figure 15 shows a block diagram of the servo processor mechanization. The oscillator for LVDT excitation, amplifier for servo loop closure, and all LVDT demodulation and monitoring are still accomplished with analog circuitry. Functions that are under program control of the servo processor microprocessor (μP) are:

1. Configuration management (active on-line operation in channels where applicable)
2. All decisions involving a logical operation
3. Control of transmission and reception of all serial bus information
4. Monitoring all serial buses for a bus failure
5. Control processor command and status storage
6. Command voting and determination of servo command to be used
7. Other servo processor's status storage
8. Failure detection within the servo processor
9. Servo modeling and comparison to actual electrohydraulic servo valve (EHSV)
10. Controlling the bypass driver of the servo processor

Design Trade-offs

Use of μPs in the Servo Processor. In previous configurations, the tasks were simple enough that a μP's extra complexity and cost could not be justified. However, the added interface capability required in this configuration, coupled with the requirement of command input voting at each servo processor, translated the required functions to a domain which would favor use of μPs over analog circuits since much of the added requirement was either logical or mathematical.

Dual Versus Single μP in Each Servo Processor. The concept of a circuit not being able to detect faults within itself, hence the need for parallel functions for comparison and fault detection, is applicable to this configuration. Even though the command output that will be computed in μP-2 is not used to position the servo loop, all of the monitor functions should compare with μP-1's control. If a discrepancy exists between μP-1, -2, either has the capability to bypass the actuator channel and transmit the servo processor fail status back on the control processor serial bus.

Analog Versus μP Functions. Since each servo processor must communicate with three serial buses and delay times on the serial buses should be relatively small, separate word detection circuits were used to interrupt-drive the μPs, rather than have word detectors under program control.

Reception and storage of three control processor's commands and statuses, mathematical comparison to control processor commands, and logical operations on status inputs and on control of binary data flow to and from the serial bus interface are all operations that favor a μP. Because of the large amount of data storage, comparison and voting, a μP is very well suited for this application. Here the program control and storage capability of a μP is a more effective mechanization than performing the same functions with logic elements and analog circuitry.

The LVDT oscillator, servo loop closure, demodulation of the LVDT secondaries, and monitoring the LVDT secondaries are still basically analog functions. Accomplishing these functions in the μP would require a much more complex μP program with significantly higher operation speed. The added design difficulty and minor, if any, parts count improvement favors leaving these functions to be done with analog circuitry.

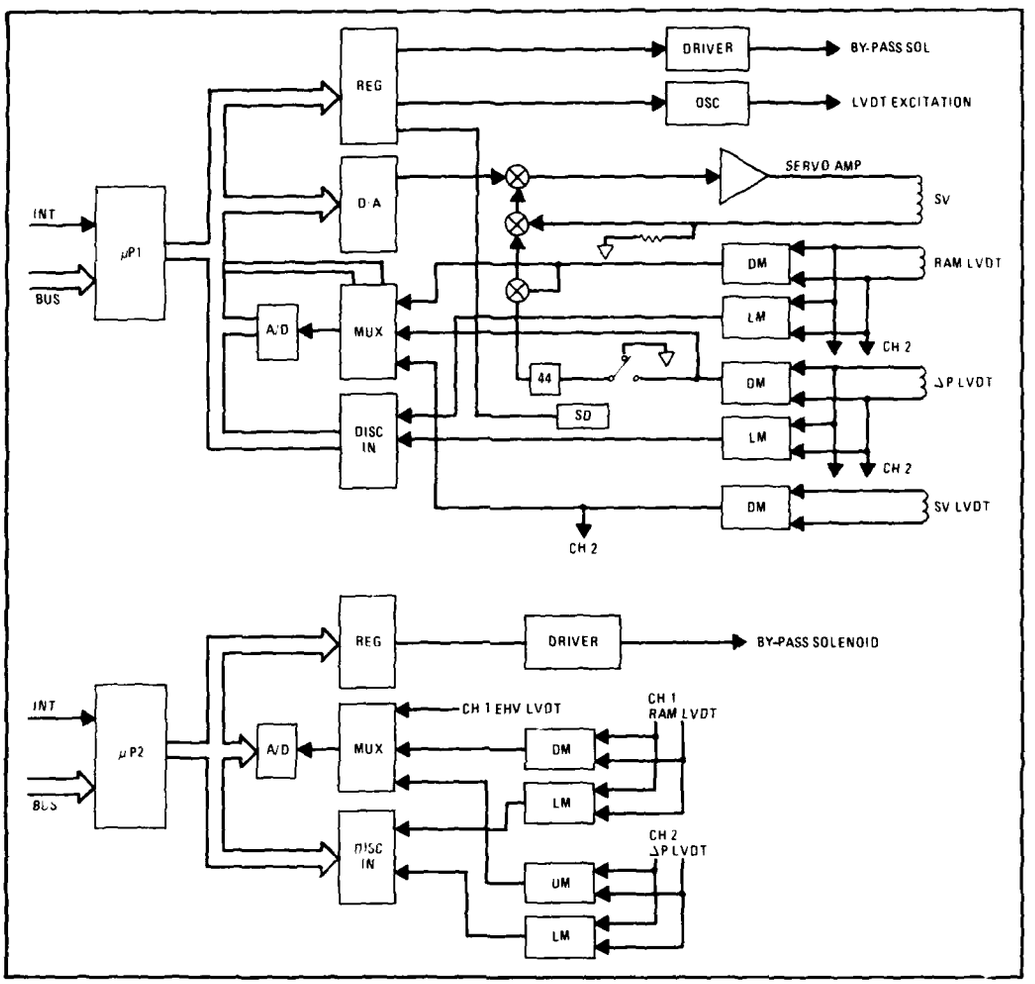


FIGURE 15. BLOCK DIAGRAM SERVO PROCESSOR MECHANIZATION

Multiplexed Analog-to-Digital (A/D) Converters Versus Dedicated. Since any incorrect data from the A/D converters into the μP will set conditions to disable the servo processor channel, the fewer parts in the A/D "feedback" the more reliable, assuming approximate equality of MTBF of the parts. Dedicated A/Ds would require three A/Ds, while using the multiplexer would require only two integrated circuits.

Smart Actuator Concept

Other factors which favor this configuration (as well as Configuration III) are its potential for providing interference-free signals to the actuators as well as providing a *"smart actuator."* This concept, by providing the actuator with a "functional completeness," yields a simpler interface between control processor and the actuation complex. Electronics integrated with the actuator allows this circuitry to provide the intelligence to compensate for mechanical non-linearities, dead zones and offsets, and to more accurately monitor performance and failures, increasing the probability of correct failure diagnosis and minimizing field test/depot test costs – a major problem at present, and potentially providing for reconfiguration upon failure or battle damage. Table I describes "Advantages of a Smart Actuation System."

TABLE 1. ADVANTAGES OF A "SMART" ACTUATION SYSTEM

- **SYSTEM PERFORMANCE**
  - IMPROVED PERFORMANCE
  - FLEXIBILITY AND MULTIMODE CONTROL AND RECONFIGURATION ON FAILURE
  - IMPROVED CHECKOUT AND SELF-MONITORING FOR FAILURE ASSESSMENT AND CORRECTION
  - COMPENSATION FOR DEGRADATION
- **EQUIPMENT**
  - MULTITECHNOLOGY APPROACH TO PRODUCT IMPROVEMENT
  - RELIABILITY ADVANTAGES OF DIGITAL CIRCUITS
    - LOW POWER CIRCUITRY
    - ARRAY DEVELOPMENT POTENTIAL
  - INTEGRATED ELECTRONIC FAILURE DETECTION CIRCUITS
  - ORDER-OF-MAGNITUDE REDUCTION IN WIRING
  - MANAGEABLE EMI, EMP CONFIGURATION
- **PROCUREMENT**
  - SIMPLIFIED INPUT-OUTPUT SPECIFICATION
  - REDUCTION IN INTERFACE PROBLEMS
  - SIMPLIFIED ASSIGNMENT OF VENDOR RESPONSIBILITIES
- **MAINTENANCE, LOGISTIC SUPPORT OF ACTUATION SUBSYSTEM**
  - AUTORIGGING OF SYSTEM
  - SELF CALIBRATION, SELF ADJUSTMENT OF NULLSHIFTS
  - ADJUSTMENT FOR OUT-OF-TOLERANCE CONDITIONS
  - DEPOT-LEVEL DIAGNOSTICS AT FLIGHT-LINE
  - REDUCTION IN TEST EQUIPMENT

5.0 OVERALL FINDINGS AND CONCLUSIONS

Studies of the mechanization of the actuation system (consisting of servo processor, servo valve, actuator devices and communication links) employing digital technologies led to conclusions in terms of architectural issues and configurations. Figure 16 demonstrates the process of digitizing the actuation system equipment elements. As this process progressed, architectural issues were analyzed and evaluated. Figure 17 presents a summary of these issues for present day and for future digital technology systems.

Physical Partitioning. Physical differences between configurations can be noted in Figure 18 and Table 2. These consist of: (1) replacing group A aircraft wiring by digital serial buses, thus saving up to 450 pounds of weight, and (2) packaging servo processors in groups of 3 instead of 12, thus reorganizing digital serial bus routing to all actuator locations and requiring a separate bus T/R unit and power supply for each servo processor.

Serial Data Bus Format, Protocol and Application. Table 3 shows a comparison among configurations of the percentage of bus utilization for condition (a) where redundancy management is performed in the servo processor or (b) where it is performed in the control processor. The MIL-STD-1553B serial data bus has limitations for control loop applications. (1) Its protocol requires high overhead-to-data word ratios (approximately 1:1) creating high utilization rates. For the selected Fault-Tolerant System, it was necessary to use the broadcast mode, thus losing its checking function. (2) Hardware is too complex, creating volume and temperature problems for remotely located servo processor applications. A revised MIL-STD serial data bus for control applications is recommended.

Functional Partitioning. It was found that the percent of bus utilization increased dramatically when the redundancy management function for the servo processor/actuator was performed in the control processor.

Advanced fighter aircraft require the implementation of additional algorithms for flight control and a high degree of integration of flight control and engine controls. This vehicle management function for the flight control computer and its interaction with the mission management computer can be facilitated if redundancy management and loop-closure functions can be performed in distributed equipments.

High Temperature Electronics. This technology involves three developments, all of which are now developed but in a non-production status: (1) I^2L semiconductor technology capable of 10^{-9} system failure rates of 200°C, (2) high temperature packaging employing eutectically mounted semiconductor chips in a hybrid microelectronics packaging concept, and (3) digital switching circuit design concepts to avoid internal heat generation.

Digital Servo Actuation Elements. In general, it was found that digital-to-analog conversions are most efficiently performed (weight, volume, cost) at the lowest possible power (preferably signal level) and simplest mechanical levels.

Redundancy Architecture. The active on-line system analyzed in these configuration lends itself to a concept of in-line failure detection throughout. Distributed servo processors are able to effectively monitor the failure status of the control processor, the interconnecting data bus, and the servo actuator complex. Failure and reconfiguration communications among channels can be readily handled by servo processors. Differences in redundancy and in battle damage reconfiguration potential show up in the configuration of power supplied, data buses, serial bus transmitter-receivers, and physical dispersion of equipment.

CONFIGURATION SELECTION. Configurations which combine servo processors with the actuator create the highest potential for future development simplicity for fighter aircraft to achieve reliabilities requiring the fewest number of channels and highest reconfiguration potential. These combinations of servo processors and actuators are termed "smart actuators" and a summary of their advantages was presented in Table 1.

The Digital Fault-Tolerant System, with a voting plane at the servo processors, has the highest potential for fault-tolerance and reconfiguration, but requires the most hardware and the highest serial bus traffic.

ACKNOWLEDGMENT

This work is a consolidation of a report AFWAL-83-3041 written by the author when employed at HR Textron and funded under Contract F33615-80-C-3623, program element 62201F, project 2403, work unit 24030275. Mr. Duane Rubertus was contract monitor for the Air Force Wright Aeronautical Laboratories.

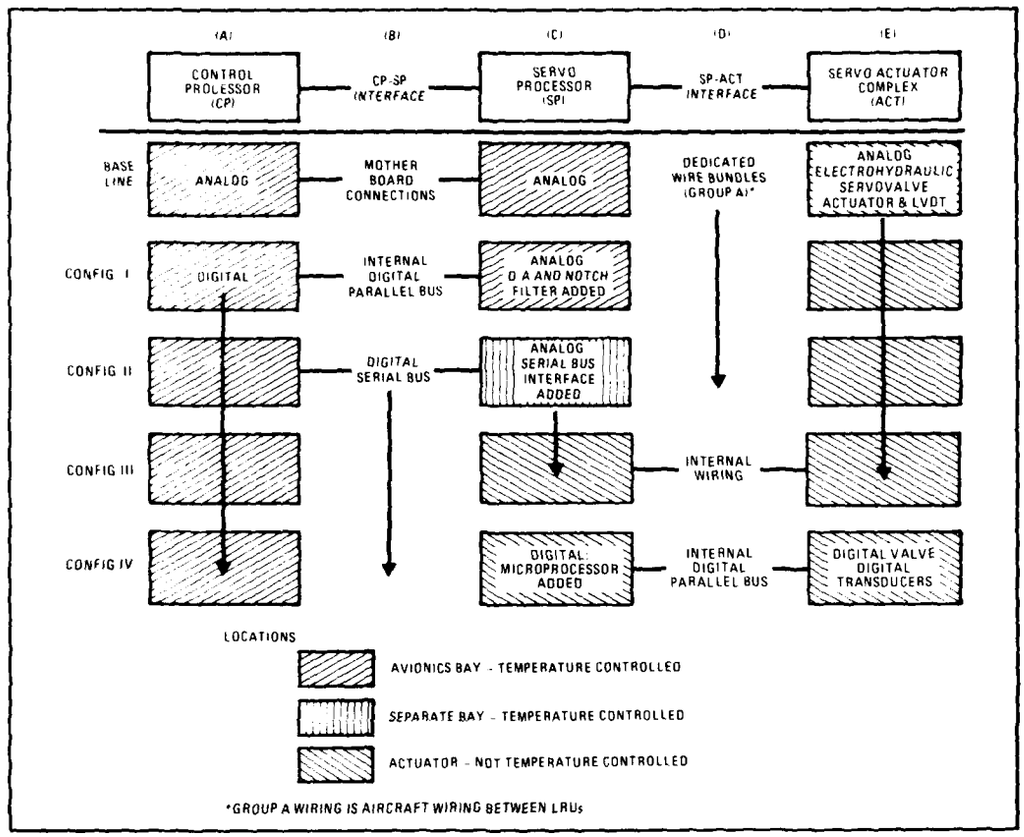


FIGURE 16. PROGRESSIVE DIGITAL CONVERSION OF ELEMENTS A-E

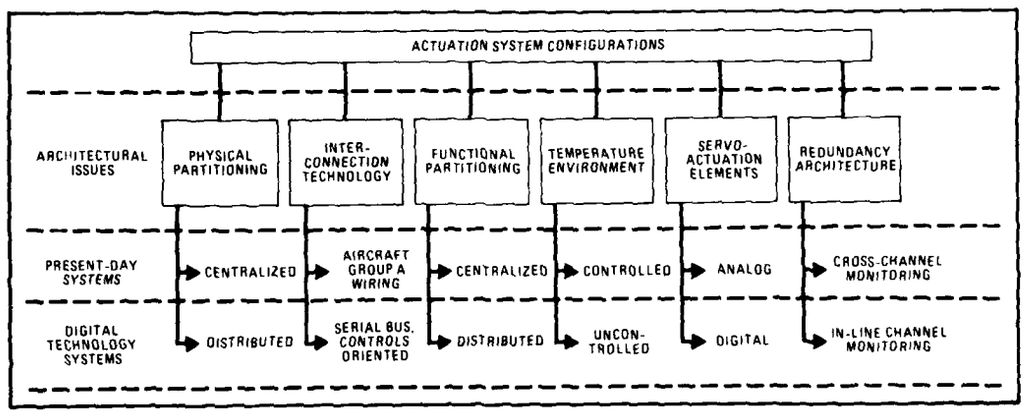


FIGURE 17. ARCHITECTURAL ISSUES

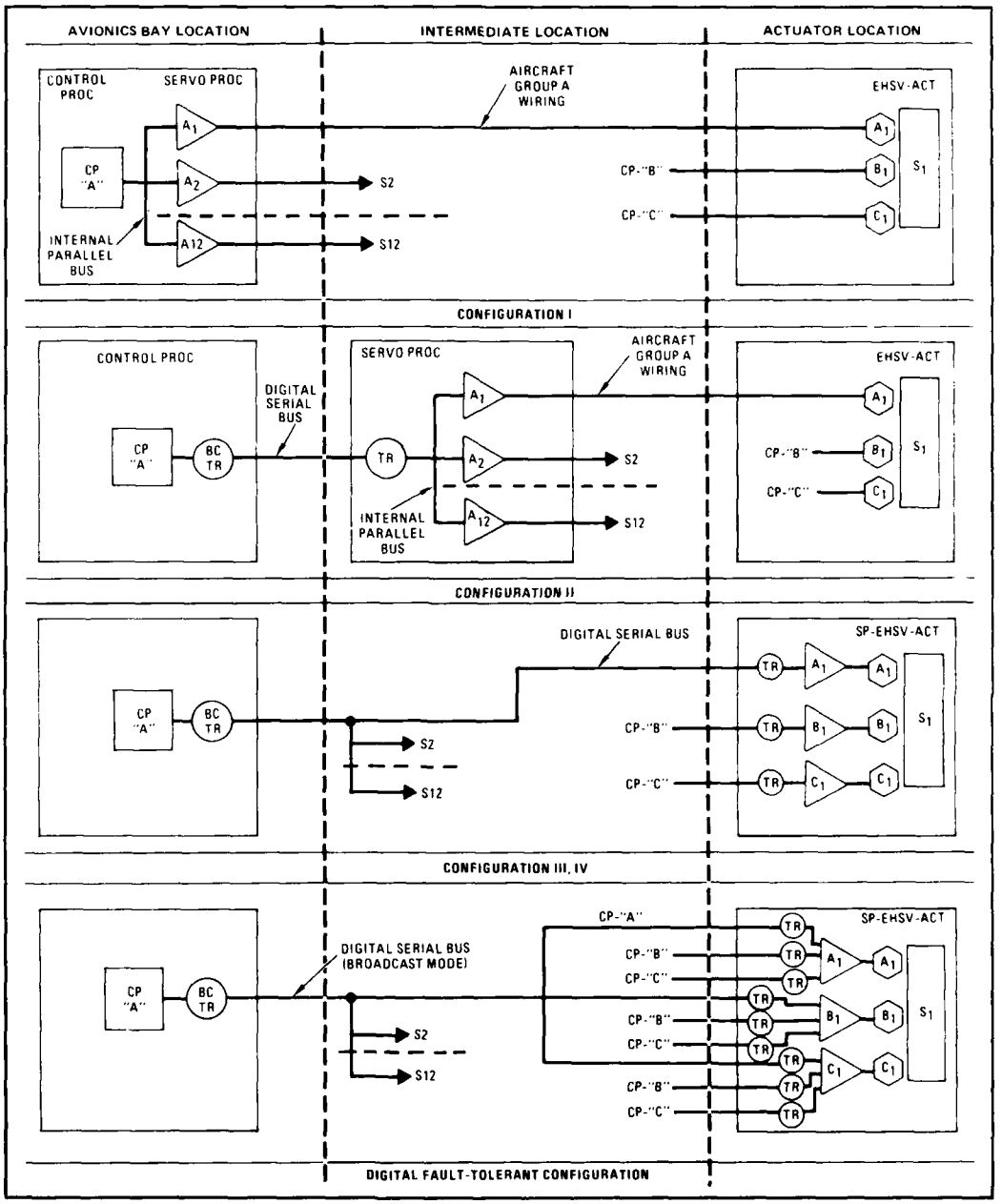


FIGURE 18. CONFIGURATION ELEMENT PHYSICAL LOCATIONS AND BUSES

TABLE 2. CONFIGURATION PHYSICAL COMPARISONS

	CONTROL PROCESSORS	CP-SP INTERCON	SERVO PROCESSORS	SP-ACT INTERCON
BASELINE	3 ENCLOSURES 3 CP'S 3 POWER SUPPLIES (LARGE)	120 MOTHERBOARD INTER-CONNECTIONS (IN CP ENCLOSURE)	36 SP'S (IN CP ENCLOSURES)	576 AIRCRAFT GROUP A WIRES FOR A 12 SURFACE SYSTEM
CONFIG I	SAME AS BASELINE	3 PARALLEL DIGITAL BUSES (IN CP ENCLOSURE)	ADD NOTCH FILTERS & D/A CONVERTORS TO 36 SP'S (IN CP ENCLOSURE)	SAME AS BASELINE ↑
CONFIG II	3 ENCLOSURES 3 CP'S 3 POWER SUPPLIES (SMALL) 3 BUS CONTROLLER/TR UNITS	3 SERIAL DIGITAL BUSES	3 ENCLOSURES 36 SP'S 3 POWER SUPPLIES (LARGE) 3 PARALLEL BUS SYSTEMS 3 TR UNITS	SAME AS BASELINE (REDUCED LENGTH OF GROUP A AIRCRAFT WIRING)
CONFIG III AND IV	SAME AS II ↑	3 SERIAL DIGITAL BUSES	12 ENCLOSURES 36 SP'S 36 POWER SUPPLIES (SMALL) 36 TR UNITS	SHORT HARNESS WIRING, SAME NUMBER AS BASE-LINE (SAVES 450 LBS OF GROUP A AIRCRAFT WIRING)
DIGITAL FAULT-TOL CONFIG	SAME AS II ↑	3 SERIAL DIGITAL BUSES	12 ENCLOSURES 36 SP'S 36 POWER SUPPLIES (SMALL) 108 TR UNITS	SAME AS III ↑

TABLE 3. SERIAL BUS UTILIZATION

		CP←SP			SP←CP			TOTAL NO.		BUS UTIL (%)
	CONFIG	NO. TRANS	NO.	DATA WORDS CONTENT	NO. TRANS	NO.	DATA WORDS CONTENT	NO. TRANS	NO. DATA WORDS	TRANS WDS TOTAL
A	II	1	1 12	CP FAIL STATUS POSITION CMDS	1	2	SP FAIL STATUS	2	15	2.2 + 3 = 5%
B	II	1	1 12	CP FAIL STATUS POSITION CMDS	4	128	RM INPUTS	5	141	5.4 + 28.2 = 34%
A	III AND IV	12	12 12	CP FAIL STATUS POSITION CMDS	12	12	SP FAIL STATUS	24	36	25.9 + 7.2 = 33%
B	III AND IV	12	12 12	CP FAIL STATUS POSITION CMDS	12	12 96	SP FAIL STATUS RM INPUTS	24	132	25.9 + 26.4 = 52%
AX	DIGITAL FAULT-TOL CONFIG	12	12 12 12	RESET POSITION CMD CP FAIL STATUS	12 SP"A" 12 SP"B" 12 SP"C"	12 12 12	SP"A" FAIL STATUS SP"B" FAIL STATUS SP"C" FAIL STATUS	48	72	51.8 + 14.4 = 66%

A – CONFIGURATION MANAGEMENT IN SERVO PROCESSOR
B – CONFIGURATION MANAGEMENT IN CONTROL PROCESSOR
X – REQUIRES USE OF BROADCAST MODE

[illegible] VALIDATION OF FLY-BY-WIRE FLIGHT [illegible]

Mary L. Hartmann
Joseph R. Wall, Jr.
Edward R. Kung

Honeywell Systems and Research Center
2600 Ridgway Parkway
Minneapolis, Minnesota
USA

SUMMARY

This paper addresses the problem of design validation of fault tolerant architectures. Finite-state machines are used to formally specify flight control functions. Their application is not new to engineering practice in flight control. However, it is believed that their systematic and formal use to form the structure of the system specification will be an aid in the design phase and in the validation phase. Examples are used to illustrate their application to flight control specifications.

The second portion of this paper is concerned with the problem of testing highly reliable systems. Models based on fault-trees in the early definition phase of estimating reliability are used to design tests to be performed at the "iron bird" state (hardware in-the-loop). Confidence in the overall system reliability is derived from a combination of component life-tests and a careful evaluation of the faults that the system is designed to accommodate without loss of control.

1.0 INTRODUCTION

The design of a fault-tolerant flight critical system is based on three important assumptions:

- All of the algorithms of the system are correctly designed and will be correctly implemented by the software and hardware of the systems.
- All possible failure modes of the components are known.
- All of the possible interactions between the system and its environment have been foreseen.

These assumptions separate an abstract concept from physical reality. There is no way that all possible failure modes of a complex system can be identified. Nor can all possible interactions of the system with its environment be modeled. Only years of experience with actual systems can instill confidence that the abstraction comes close to physical reality.

A distinction is often made between verification and validation. Verification applies to software, it is the process of demonstrating that software is technically correct by showing that each software function performs as specified and the technical aspects of inputs, outputs and the passage of data between functions are correct. It must be shown that the data that defines the state of the function are not corrupted by any side effect so that the data survive to the subsequent cycles of the calculations. Validation applies to the system, it is the process of showing that the system performs according to its requirements and reacts favorable in all situations.

The importance of verification grows proportionately with the trend to delegate larger percentages of the system functions to software. As the software becomes more complex or more critical, verification must become more systematic and formal and must establish complete confidence in the performance of the software.
The problem of system validation is divided into two parts:

- Show that the system does indeed behave as our mathematical model is formulated for all normal functions of the system and for all classes of hypothesized failures
- Estimate how closely the mathematical model abstracts reality

Formal validations address the first point. To be able to do this with any certainty, one must have a precise specification of the normal functions that the system must perform and a precise description of the classes of failure events to which the system must respond. Any lack of precision in these specifications will result in a lack of certainty and confidence in the validation. In order to carry out a meaningful validation, a systematic methodology is required for specifying, designing, and implementing the flight control system. Such methodology helps to make the validation process clear throughout design and development and test stages.

Although a verification and validation methodology proves many functions, there are several that it cannot prove. It is not possible to predict the response of a computer to all possible failures of its hardware. The most elaborate failure-and-effects analysis can only enumerate the most probable failure modes. Because there is a vast number of states in which the computer may be when a failure occurs, the outcome can only be estimated. The system must be configured so that there are no failures that can put the airplane in jeopardy. Proofs can only show that the system is safe against classes of hypothetical mathematical failures, only experience can show that physical failures are covered.

The remainder of this paper addresses two related areas. Section 2 discusses formal methods for specifying system functions and presents three flight control examples. Section 3 presents some ideas for validating system reliabilty; in particular for highly redundant systems where life-cycle tests are too time consuming.

2.0 FORMAL METHODS FOR SPECIFYING SYSTEM FUNCTIONS

Formal validation is predicated upon a methodology for specifying the functions of the system, designing to capture these specifications, and implementing the resultant design. A methodolgoy must provide sufficient precision in order to carry out a formal validation. It should provide guidance for testing to validate the total integrated system. Moreover, it serves to keep clear the importance of validation throughout the design process.

The view of the NASA Working Group on Validation Methods for Fault-Tolerant Avionics and Control Systems (Ref. 1) support this requirement.

> "In order to make a rigorous case that a fault-tolerant system is valid embodiment of its requirements, a systematic approach is required that is closely tied to the design process."

The terms "requirements" and "specifications" are often not rigorously defined. "Requirements" generally mean the informal statements about the functions and performance of a system. These are prepared by the customer and may not be precise or complete. The specifications, or requirements specification, are the documents that try to capture the requiremens in a more formal manner. These must be as complete and precise as needed to ensure the success of a project. Indeed, many errors are made in obtaining the correct description of what a system is supposed to do. The final validation of a system returns to these specifications.

In converting from requirements to specifications, there are varying degrees of formality. These range from formal languages like SRI's SPECIAL (ref. 2) to documents prepared according to various military standards (ref. 3). The following section examines one method of specifying flight control functions that is attractive.

2.1 Functional Descriptions

Any information or signal processing system may be thought to be made up of two flows--one is the information or data being procesed by the system, the other is the sequence of control actions that manipulates the data (ref. 4).

Petri nets (ref. 5) and LOGOS (ref. 6 and 7) are two graphical techniqes for describing flows. A Petri net is a directed, bipartite graph of alternating vertices called places and transitions. It provides an abstract model of information and control flows. The major applications of Petri nets have been in systems in which some of the events occur concurrently, but with constraints on the concurrence, predecence, or frequency of the events. The graphical technique LOGOS portrays these two flows in parallel graphs. The control graph initiates, sequences, and synchronizes the data operations on the data graph. LOGOS has been used to analyze very complicated systems, including the Air Force DAIS architecture (ref. 8).

In many systems the structure for producing one of the flows is more complicated or fundamental to the system than the other. For example, in handling huge quantities of data, the organization of the data is central in designing efficient algorithms. In this case that the data flow dominates the design considerations. For flight controls, the calculations on the data are not complicated, but the structure for controlling the computations is. Control flow dominates. Thus, the design will be concerned chiefly with the control structure; the data flow will follow along naturally.

A finite-state machine is the simplest computing structure. At the next level are the push-down automata, which have stack memories. The most general theoretical computing structure is the Turing machine. Finite-state machines use two expressions, called states and events. The states correspond to the sequential circuits familiar to electronics engineer. Events represent an input to the control structure, signalling some important point of activity in its environment. This description appears to be appropriate for flight control software.

The advantage of the finite-state machine representation is that it is precise and may be easily reviewed by control engineers for completeness. It may be used to describe system-level functions; it is not limited only to hardware or software. The states must be clearly identified and the events causing transitions must be defined. This provides a structure that may be completely tested.

Fortunately, all flight control functions are either straight-line calculations requiring no past data, or calculations requiring only a fixed, finite set of past data. Hence, the latter functions may be represented as finite-state machines.

A general finite-state machine is diagrammed in Figure 1. When inputs are received, outputs are calculated as functions of the current values of the state variables and the inputs. Then the machine switches to a new state, again as a function of the current state and the input quantities. It is often useful to produce outputs associated with these state transitions; for example, a warning to the pilot upon automatic change of mode due to loss of an input signal. These representations were found to be very natural for mode switching, signal selection, synchronization, and failure management.

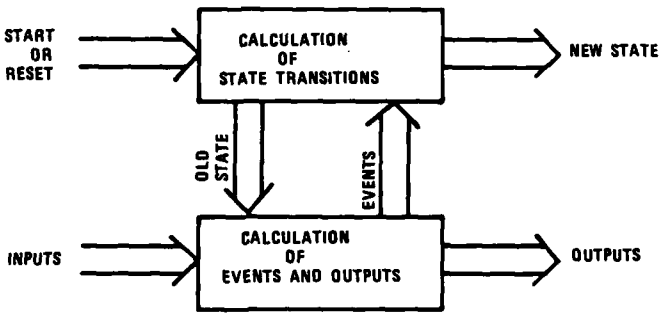


FIGURE 1 - A finite-state machine.

Finite-state machines are represeented in two different ways--as a directed graph or a table. The directed graph approach is more intuitive because it is a picture. As the number of states, events, and state transitions grows, this advantage is effectively negated by the complexity of the diagrams. States are represented in the diagram as circles; legal state transitions are displayed as directed arrows connecting one state with another. The event that triggers a particular state transition is labeled on the arrow. The diagrams are interpreted in the following way. At any time, the finite-state machine is in a current state. When an event is detected and received, the machine will make the state change indicated by the outgoing arrow labeled with that event. For a deterministic machine there can be at most one such arrow. When no such labeled arrow exists for the current state, this represents an error, and the machine will attempt to recover. The most simple recovery action is to ignore the event. The action sequence performed by the machine while changing state is generally not included in the diagram.

The alternative representation is to describe the state transitions with a table or matrix. The entries in the matrix contain the number of the new state and an ordered list of actions to be performed to effect a change in state (possibly null). Blank entries are illegal state transitions and could contain some code to assist recovery.

As the number of states and events grows larger, there is a need to partition the state machine so that each part is more manageable. To increase the clarity of the control structure, this partition should be done based on logical properties, and not in an arbitrary manner.

The functions for flight control fall into categories which fit a finite state machine description:

(1) The executive structure (initiative, branch in the rate tree, recover from power interrupts, equalize integrations, maintain the dynamic filter states, annunciate system status)

(2) Data transfers (input, output, exchange data between channels)

(3) Control mode switching and dynamical switching within control modes

(4) Control law calculations (outer loops, inner loops, gain schedules)

(5) Synchronization (synchronize channels, time-synchronize programs for transfers, etc)

(6) Built-in-test functions (preflight checks, on-line checks)

(7) Selection from redundant input signals

(8) Failure detection and reconfiguration

It is also necessary to show a global consistency between these functions, particularly the built-in-tests and the failure management functions.

Some examples are presented in the following section.

2.2 Flight Control Examples

This section illustrates the use of the finite state machines in specifying flight control functions. Three examples are presented in the following paragraphs. The first application is taken from the Demonstration Advanced Avionics Systems (DAAS), an experimental digital system for general aviation flight-tested at NASA-Ames Flight Research Center. This example use machines to describe the mode logic.

The second example treats interchannel communication and cross-channel voting in a conventional, frame synchronous, triple channel computer system. Finite state analyses is used to analyze the fault detection performance by examining the effects of various component failures.

The final example uses finite-state machines to specify the functions to be performed in a remote actuation terminal. The terminal positions control surfaces based on commands received from redundant computer channels. The remote terminal performs signal selection on the incoming commands and management of the hydraulic valves and associated electronics.

2.2.1 Example 1 -- Specification of the DAAS Flight control Requirements

The Demonstration Advanced Avionics System (DAAS) provided an excellent opportunity to apply finite-state structures for specification and design in an engineering environment. The system is not complicated, yet, it illustrates many of the problems in precisely specifying flight control software.

DAAS provides the following functions:

- Autopilot/Flight Director
- Navigation and Flight Planning
- Flight Warning System
- Operating limits data
- Communication and Identification
- Engine Instrumentation
- Weight and Balance Data
- Normal and Emergency Procedural Checklists
- Weather Avoidance
- Built-in self Diagnostic Tests

The states are defined by the services being provided. Six states are allowd by the system's requirements:

0. Flight director off, yaw damper off, autopilot off
1. Flight director on, yaw damper off, autopilot off
2. Flight director off yaw damper on, autopilot off
3. Flight director on yaw damper on, autopilot off
4. Flight director on yaw damper on, autopilot on
5. Flight director on, yaw damper on, autopilot on, control wheel steering on

In addition to the cockpit, switches there are validity signals from the sensors and from the software monitors for the computer system and the pitch trim system.

All of the normal events that can affect the six system states must act through the following switches and flags:

1. Flight director switch
2. Yaw damper paddle switch
3. Autopilot paddle switch
4. Control wheel steering switch
5. Go-around switch
6. Manual electric trim switch
7. Autopilot dump switch

The entries in tabel 1 show the number of the state to which the system will switch when the corresponding event occurs. Most of the transitions listed in the table are trivial. For example, if the system is in State 2 with the flight director off and the flight director switch is turned on, the system switches to State 3, honoring the request. Some of the entries are not active; if the yaw damper is not on, the event of switching if off cannot occur. But there are a few that are not trivial. These represent decisions for the requirements. For example, with everything off in State 0, the go-around switch or the control wheel switch will turn on the flight director. Note that in States 4 or 5 with the auto-pilot on, the event of switching off the flight director is ignored.

Table 1 will be transformed into the requirements for software by taking into account the details of the hardware mechanisms.

The requirements of the system's behavior at this top-level abstraction are represented precisely and completely by the machine of Table 1. One step in the verification will be shown that the software-plus-hardware mechanization correctly implements this machine. This table will be part of the basis for the final validation of the system.

TABLE 1 **SYSTEM FINITE-STATE MACHINE**

STATE \ EVENT	autopilot dump switch off	manual electric trim switch on	go-around on	control wheel switch on	control wheel switch off	flight director on	flight director off	yaw damper on	yaw damper off	autopilot on	autopilot off
0. servos off flight director off	0	0	1	1	0	1	-	2	-	0	-
1. servos off flight director on	0	1	1	1	1	-	0	3	-	1	-
2. yaw damper on autopilot off flight director off	0	2	3	3	2	3	-	-	0	2	-
3. yaw damper on autopilot off flight director on	0	3	3	3	3	-	2	-	1	4*	-
4. yaw damper on autopilot on flight director on	0	3	3	5	-	-	4	-	-	-	3
5. yaw damper on, autopilot on, flight director on, control wheel steering on	0	3	3	5	4*	-	5	-	-	-	3

*this transition uses a 2.5 second fade-on ramp to avoid abrupt action. Starting State: State 0.

The requirements reflected by Table 1 must be captured by a combination of hardware and software functions. In this system, the requirement that the autopilot can be on only if the yaw damper is on was accomplished by mechanically linking the two switches so that one cannot be turned on without the other. The requirement that the autopilot cannot be on without the flight director must be enforced in the software. The switch for the flight director is momentary-contact and alternate activations are interpreted as alternate requests for on and off. These considerations lead to Table 2 which give the transitions that were implemented in the software.

Reference 9 further describes the mode logic development. The computations that drive the flight director and autopilot are specified with two additional finite state machines which further detail 14 lateral modes (states) and 9 pitch modes (states). Use of these finite machine tables was found to be an excellent way to make all design decisions visible and prevent errors of omission. A multi-microcomputer implementation based on INTEL 8086 hardware was successfully tested on a Cessna 402B aircraft at NASA/Ames Research Center.

TABLE 2 **SYSTEM MACHINE IN SOFTWARE**

STATE \ EVENT	autopilot dump switch on	manual electric trim switch on	go-around on	control wheel switch on	control wheel switch off (released)	flight director button on	yaw damper on	yaw damper off	autopilot on	autopilot off	yaw damper and autopilot on	yaw damper and autopilot off
0. servos off flight director off	0	0	1	1	0	1	2	-	-	-	0	-
1. servos off flight director on	0	1	1	1	1	0	3	-	-	-	4*	-
2. yaw damper on autopilot off flight director off	0	2	3	3	2	3	-	0	2	-	-	-
3. yaw damper on autopilot off flight director on	0	3	3	3	3	2	-	1	4*	-	-	-
4. yaw damper on autopilot on flight director on	0	3	3	5	-	4	-	-	-	3	-	1
5. yaw damper on, autopilot on, flight director on, control wheel steering on	0	3	3	5	4*	5	-	-	-	3	-	1

*this transition uses a 2.5 second fade-on ramp to avoid abrupt action. Starting State: State 0.

2.2.2 Example 2 -- Cross-channel voting and testing of interchannel communications

This section describes the interchannel communication typical of a frame-synchronized triplex system. The configuration is shown in Figure 2. Each computer communicates to the others through a single transmitter, which sends the same signals to receivers at each of the other computers. It is assumed that the sending computer-transmitter cannot originate two different signals. Asymmetry in the communications can be caused only by errors in the receivers or the receiving computer. This assumption must be justified by a failure mode and effects analyses. Under different assumptions (reference 10), four computers are needed to detect one error if the originating computer sends different signals to the others. This is not the case for the configuration shown in Figure 2.

Assume that any one of the 12 elements in Figure 2 produces errors and then follow these errors through two levels of data exchange. Only one unit is assumed faulty. Errors are detected by a sum check on the data transmissions and by comparisons of computer outputs from some active computation. The error syndromes after the initial data exchange are listed in table 3, the final syndromes allow a computer to detect errors in the foreign computers or the communications channels, but cannot distinguish between errors in the computers, transmitters, or receivers. After the second round of data interchange, the syndromes distinguish receiver errors and computer-transmitter errors; the local computer, if okay, can determine that its transmitter is causing errors.

In the second round of communications, a computer will receive word that indicates an error in the left or right path, or its own transmitter. The transmission over an erroneous path is indicated by an X in table 4.

The algorithm is summarized in table 5. There is a jump in the frame of reference from the initial observation to the final analysis in table 3, if the right channel decides that its left channel is in error, then the local channel will interpret this decision to mean that it is in error.

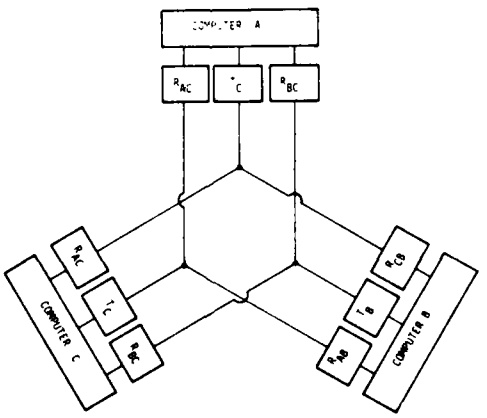


FIGURE 2 - **Communication among synchronized channels.**

TABLE 3 - INITIAL FAULT OBSERVATIONS

EVENT / ERROR IN / OBSERVATION	SUM CHECK ON C TO A	SUM CHECK ON B TO A	COMPARISON CB AT A	COMPARISON AC AT A	COMPARISON BA AT A	CONCLUSION OF COMPUTER A	SUM CHECK ON A TO B	SUM CHECK ON C TO B	COMPARISON AC AT B	COMPARISON BA AT B	COMPARISON CB AT B	CONCLUSION OF COMPUTER B	SUM CHECK ON B TO C	SUM CHECK ON A TO C	COMPARISON BA AT C	COMPARISON CB AT C	COMPARISON AC AT C	
1 C_A	X	X	X	X	X	PROBABLY SOMETHING WRONG	X	OK	FAIL	FAIL	OK	ERROR IN A TO B IN COMPUTER A	OK	X	FAIL	OK	FAIL	ERROR IN A TO C OR IN COMPUTER A
2 T_A	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN A TO B OR IN COMPUTER A	OK	FAIL	X	OK	X	ERROR IN A TO C OR IN COMPUTER A
3 R_{AB}	OK	FAIL	X	OK	X	ERROR IN B TO A OR IN COMPUTER B	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM
4 R_{CA}	FAIL	OK	X	X	OK	ERROR IN C TO A OR COMPUTER C	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM
5 C_B	OK	X	FAIL	OK	FAIL	ERROR IN B TO A OR COMPUTER B	X	X	X	X	X	PROBABLY SOMETHING WRONG	X	OK	FAIL	FAIL	OK	ERROR IN B TO C OR IN COMPUTER B
6 T_B	OK	FAIL	X	OK	X	ERROR IN B TO A OR COMPUTER B	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN B TO C OR IN COMPUTER B
7 R_{CB}	OK	OK	OK	OK	OK	NO PROBLEM	OK	FAIL	X	OK	X	ERROR IN C TO B OR IN COMPUTER C	OK	OK	OK	OK	OK	NO PROBLEM
8 R_{AB}	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN A TO B OR IN COMPUTER A	OK	OK	OK	OK	OK	NO PROBLEM
9 C_C	X	OK	FAIL	FAIL	OK	ERROR IN C TO A OR IN COMPUTER C	OK	X	FAIL	OK	FAIL	ERROR IN C TO B OR IN COMPUTER C	X	X	X	X	X	PROBABLY SOMETHING WRONG
10 T_C	FAIL	OK	X	X	OK	ERROR IN C TO A OR IN COMPUTER C	OK	FAIL	X	OK	X	ERROR IN C TO B OR IN COMPUTER C	OK	OK	OK	OK	OK	NO PROBLEM
11 R_{AC}	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM	OK	FAIL	X	OK	X	ERROR IN A TO C OR IN COMPUTER A
12 R_{BC}	OK	OK	OK	OK	OK	NO PROBLEM	OK	OK	OK	OK	OK	NO PROBLEM	FAIL	OK	X	X	OK	ERROR IN B TO C OR IN COMPUTER B

TABLE 4 - SECOND DATA INTERCHANGE

EVENT / ERROR IN / REPORT	OBSERVATION OF A	REPORT OF B TO A	REPORT OF C TO A	CONCLUSION OF COMPUTER A	REPORT OF A TO B	OBSERVATION OF B	REPORT OF C TO B	CONCLUSION OF COMPUTER B	REPORT OF A TO C	REPORT OF B TO C	OBSERVATION OF C	CONCLUSION OF COMPUTER C
1 T_A	X	X	X	X	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A
2 T_A	OK	A TO B OR C_A	A TO C OR C_A	ERROR IN T_A	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A	X	A TO B OR C_A	A TO C OR C_A	ERROR IN CHANNEL A
3 R_{BA}	B TO A OR C_B	X	OK	ERROR IN R_{BA}	B TO A OR C_B	OK	OK	ERROR IN R_{BA}	B TO A OR C_B	OK	OK	ERROR IN R_{BA}
4 R_{CA}	C TO A OR C_C	OK	X	ERROR IN R_{AC}	C TO A OR	OK	OK	ERROR IN R_{AC}	C TO A OR C_C	OK	OK	ERROR IN R_{AC}
5 C_B	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B	X	X	X	X	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B
6 T_B	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B	B TO A OR C_B	OK	B TO C OR C_B	ERROR IN T_B	B TO A OR C_B	X	B TO C OR C_B	ERROR IN CHANNEL B
7 R_{CB}	OK	C TO B OR C_C	OK	ERROR IN R_{CB}	OK	C TO B OR C_C	X	ERROR IN R_{CB}	OK	C TO B OR C_C	OK	ERROR IN R_{CB}
8 R_{AB}	OK	A TO B OR C_A	OK	ERROR IN R_{AB}	X	A TO B OR C_A	OK	ERROR IN R_{AB}	OK	A TO B OR C_A	OK	ERROR IN R_{AB}
9 C_C	C TO A OR C_C	C TO B OR C_C	X	ERROR IN CHANNEL C	C TO A OR C_C	TO B OR C_C	X	ERROR IN CHANNEL C	X			
10 T_C	C TO A OR C_C	C TO B OR C_C	X	ERROR IN CHANNEL C	C TO A OR C_C	TO B OR C_C	X	ERROR IN CHANNEL C	C TO A OR C_C	C TO B OR C_C	PO	ERROR IN T_C
11 R_{AC}	OK	OK	A TO C OR C_A	ERROR IN R_{AC}	OK	OK	A TO C OR C_A	ERROR IN R_{AC}	X	OK	A TO C OR C_A	ERROR IN R_{AC}
12 R_{BC}	OK	OK	B TO C OR C_B	ERROR IN R_{BC}	OK	OK	B TO C OR C_B	ERROR IN R_{BC}	OK	X	B TO C OR C_B	ERROR IN R_{BC}

TABLE 5 **SUMMARY OF FAILURE ANALYSIS ALGORITHM**

SUN CHECK RIGHT TO LOCAL	SUN CHECK LEFT TO LOCAL	COMPARISON OF RIGHT AND LEFT	COMPARISON OF LOCAL AND RIGHT	COMPARISON OF LEFT AND LOCAL	INITIAL FAULT OBSERVATION
X	OK	FAIL	FAIL	OK	RIGHT CHANNEL
FAIL	OK	X	X	OK	RIGHT CHANNEL
OK	FAIL	X	OK	X	LEFT CHANNEL
OK	X	FAIL	OK	FAIL	LEFT CHANNEL
OK	OK	OK	OK	OK	OK
ALL OTHERS					LOCAL CHANNEL

(X = OK OR FAIL)

INITIAL FAULT OBSERVATION

REPORT FROM THE RIGHT	OBSERVATION OF LOCAL	REPORT FROM THE LEFT	FAULT DIAGNOSIS
OK	OK	OK	OK
X	RIGHT	RIGHT	RIGHT CHANNEL
LEFT	LEFT	X	LEFT CHANNEL
LOCAL	OK	LOCAL	LOCAL TRANSMITTER
LOCAL	OK	OK	LOCAL TO RIGHT RECEIVER
LEFT	OK	OK	RIGHT TO LEFT RECEIVER
OK	OK	RIGHT	LEFT TO RIGHT RECEIVER
OK	LEFT	X	LEFT TO LOCAL RECEIVER
X	RIGHT	OK	RIGHT TO LOCAL RECEIVER
ALL OTHERS			LOCAL CHANNEL

(X = OK OR LEFT OR RIGHT OR LOCAL OR ERROR)

SECOND DATA INTERCHANGE

2.2.3 Example 3 -- Function of the Remote Actuator Terminal

Three hydraulic cylinders are connected to sum forces to drive an aerodynamic surface and provide triple redundancy. Any cylinder alone can position the surface. Each cylinder has a solenoid-held engage/bypass valve to control the status of the cylinder. The position of the cylinder, the position of the spool of the Electro-Hydraulic Servo Valve (EHSV), and the pressure differential across the cylinder are measured by Linear Variable Differential Transformers (LVDTs). Each of these feedback sensors provides a signal attesting to the validity of the LVDT output.

Redundant servo commands are serially transmitted to the remote terminal from multiple computing channels. These will carry parity bits with which the fidelity of the transmission and the status of the sending channel may be determined.

The remote terminal must select a suitable signal from the computer channels and position the surface according to this command. The remote terminal must be operational following any two like-component failures. The failure may be in mechanical, hydraulic, or electrical components.

Hierarchy of Functions. - The top-level function may be further specified in terms of the lower-level functions that are necessary. Figure 3 illustrates this decomposition. Design decisions are made in constructing this decomposition.

Drive EHSV according to comand. - One of second-level functions of the remote terminal is to drive the EHSV of each redundant channel. This is accomplished by obtaining the input servo command and computing the control value for the EHSV. The serial digital transmissions from multiple computing channels must be received and interpreted. The presence of a signal and the validity of the transmission must be determined by the subfunction "validate signal transmissions." The "select command" subfunction must choose from among the valid signals (perform median, averaging, or some selection process). The surface position, the valve spool position, and the differential of pressure in the cylinder are fed back and combined with the selected command as specified by the servo system control law. This control law computation may be analog or digital. An analog signal to drive the EHSV must be provided.

Redundancy management of servo channels. - The other second-level function of the remote terminal is to perform the redundancy management of the servo channels. Redundancy management includes:

(1) Monitoring the health of each channel,
(2) Either engaging or bypassing a channel based on its health, and
(3) Relieving the force fight among the engaged channels.

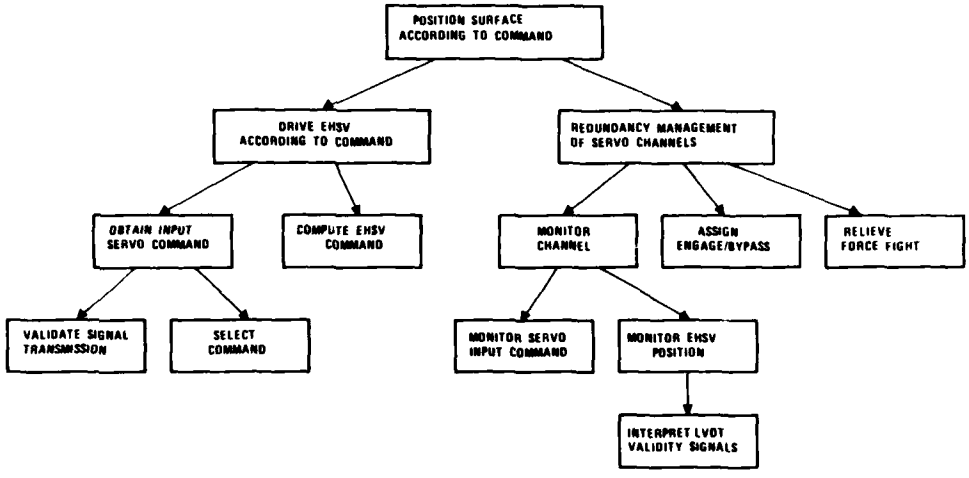


FIGURE 3 - **Hierarchy chart of specifications for the remote terminal.**

The monitor channel subfunction is to provide the failure detection mechanism for the general operation of the servo channel. It may require self-checking circuitry. Proper monitoring of any D/A or A/D translations is required. A comparison between expected valve-spool position as predicted by a model and the measured valve-spool position is suggested. A subfunction to interpret LVDT validity signals is required to determine that the feedback sensors are all functioning properly. Each channel must be engaged or bypassed on the basis of the output of the monitor channel subfunction. Also, a mechanism must be included to relieve the force fight among the force-summed cylinders. The use of an active/on-line assignment with pressure differential feedback is suggested.

Finite-state machine description. - The redundancy management of the three servo channels can be specified as a finite-state machine. The health monitoring function of each channel is used to assign an engage or bypass status. Each servo channel can be in one of three possible states:

(1) Engaged and active
(2) Engaged and on-line
(3) Bypassed

There are $3^3 = 27$ states; the various transitions can be described based on changes in engage/bypass or active/on-line status. In order to study the engage/bypass function it is useful to cluster the 27 states into the eight groups shown in Figure 4. Here transitions occur only if the engage status of any channel changes. States within each cluster cover all the active/on-line assignments, including those resulting from logic failures (i.e., all active or all on-line). To verify this function in the remote terminal, it must be shown that all transitions between clusters operate as specified. For example, the event "channel A bypassed" must take any of the eight states in cluster 1 to one of the states in cluster 2. To examine the active/on-line logic, the states within a cluster can be verified, and it must be shown that the active/on-line logic does not cause transitions between clusters.

The finite-state description is useful because it requires the designer to consider all the possibilities. In addition, by clustering the states, the operation of the engage/bypass logic can be separted from the active/on-line function.

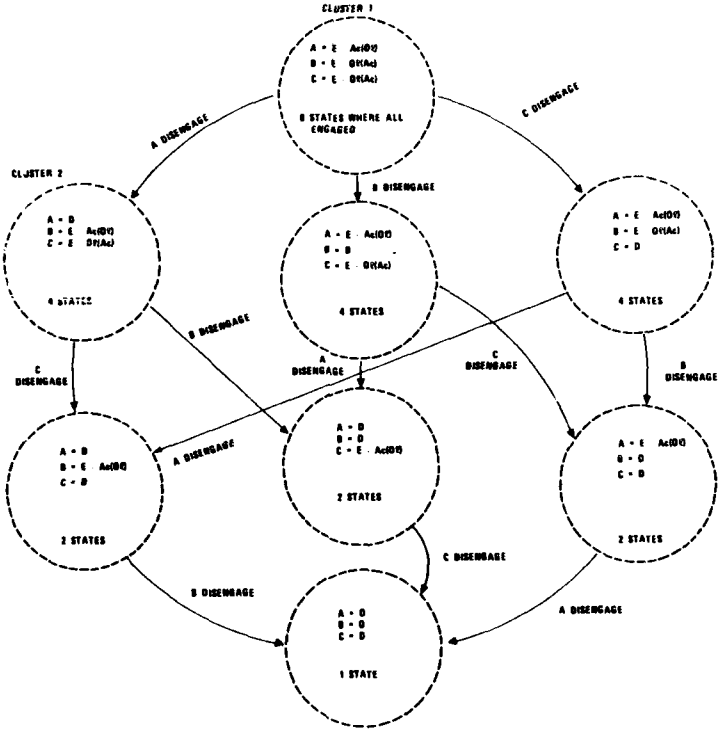


FIGURE 4 - **Finite-state model for the remote terminal.**

3.0 VALIDATION OF SYSTEM RELIABILITY

In this section, the focus changes from specifying software to one of validating the complete system (hardware and software). In flight-critical applications, a major aspect of the system validation involves predicting the reliability of the over-all system including the redundant hardware elements and their associated software to identify and manage faults.

Estimates of reliability are obtained during the process of defining the flight control architecture. Once a candidate architecture has been defined, a detailed analysis to estimate the probability of loss of control per hour of flight must be performed. This theoretical estimate must be substantiated with laboratory tests.

Validation by testing can be divided into two areas, as shown in Figure 5. In the area of components and subsystems, it is feasible to run life tests to statistically validate mean time before failure (MTBF) predictions. During the system test phase, it is not possible to run life tests. Instead, integrated system (iron bird) tests are used to verify the fault tolerance achieved by redundancy. These results, when combined with the component MTBFs, permit extrapolation of the complete systems's reliability.

Section 3.1 discusses the design of test cases for integrated system testing. The finite-state machines used to specify the functions are now used in the validation testing phase. The fault tree model is used to identify the various combinations of faults that need to be tested to demonstrate the predicted fault tolerance.

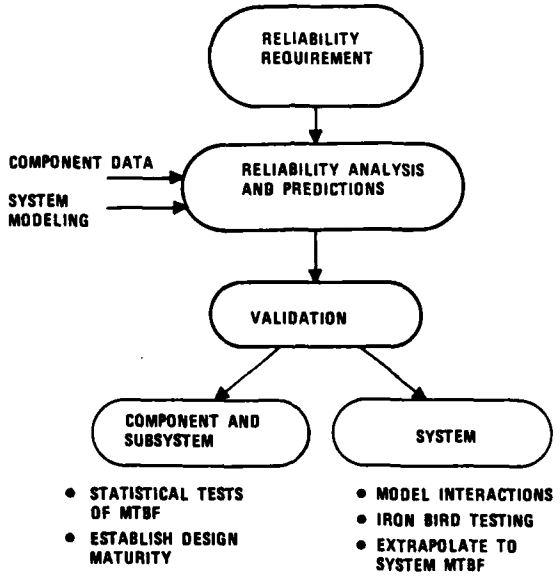


FIGURE 5 - **The validation of an ultra-reliable system depends on indirect testing.**

3.1 Design of Test Cases -- Ultra-reliability of flight-critical systems is achieved through redundant hardware. The system must be able to tolerate multiple faults while maintaining undegraded flight operation. Validation of the fault-tolerance and reconfiguration features is the most critical step to the validating the reliability of the total system. These processes can best be accomplished in an iron bird, in which a high degree of fidelity to the flight environment is obtained by including actual flight hardware operating in a real-time, closed-loop simulation. The fundamental problem of fault tolerance validation is the vast number of test cases when all possible combinations of flight conditions and multiple hardware faults are considered. Effective testing requires:

(1) A methodology using both theoretical and practical perspectives to define a manageable set of test cases.

(2) Automating the testing as much a practical.

Consider the set of conceptial states shown in Figure 6. It can be used to visualize possible test cases. The universe of test possibilities is first divided into two areas: everything operable and some element failed. The operative region can be described by one or more finite-state machines. The effect of various failures on the system can be described with a fault tree. The "failed element" region includes failures for which a reconfiguration strategy was designed (i.e., swithes out failed element), as well as failures external to the system which are to be tolerated (i.e., loss of a hydraulic power supply). The management of redundant elements can be described using finite-state machines.

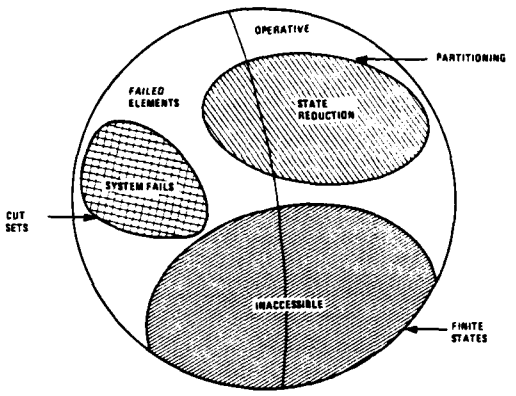


FIGURE 6 - **Conceptual system states.**

Three cross-hatched areas represent regions where analysis can be used to reduce the number of test cases:

(1) System failed - These cases, identified as cut sets in the fault tree, are discussed in more detail below.

(2) Inaccessible - These states represent combinations of modes, flight conditions, or environments that are mutually exclusive and do not need to be test cases.

(3) Partitioned - This area represents states that can be partitioned such that all combinations do not need to be tested. This could involve use of hierarchies of finite-state machines or partitioning on the basis of control modes, etc. The next section outlines method for desinging test cases based on the finite-state models.

3.2 Use of Fault-Tree Analysis - Basically, the fault tree is a top-down method of describing the failure of a system. The top event is the occurrence of total system failure, modeled by logical combinations of the failure of its associated subsystems. This process is repeated for structuring the subsystems until reaching the lowest event--the failure of the basic components. Boolean expressions are generated that list all possible minimal combinations of component faults leading to total system failure. Each of these fault combinations is known as a minimal cut set. The probability of total system failure is computed by combining the probability of occurrence for each minimal cut set.

The fault-tree analysis of a system can be used to develop test states for validating fault tolerance. The purpose of fault-tolerance testing is not to prove that the system fails when the fault tree predicts it will, but rather the converse. The purpose of this testing is to establish that the system works correctly when the fault tree predicts it will. To establish the former, the various fault combinations that make up minimal cut sets are used as test states, and the failure of the system is expected. This testing would demonstrate that the system fails at least as often as the fault tree predicts. The important case to establish is the latter. In this case, various combinations of faults that do not contain cut sets are used as test states. The system is expected to work correctly for all of these combinations. If it does, then this testing demonstrates that it works at least as often as the fault tree predicts.

A "test set" is defined as a set of component failures that contains no cut sets. This means that the fault-tree analysis predicts the system should not fail in the face of failures contained in any test set. A "maximal test" set is defined as a test set that is not contained in any test set. This means if any component failure is added to a maximal test set, the resulting set is a cut set. The relationships among these sets of component failures are shown in Figure 7.

One way of generating test sets is to consider the maximum components in each cut set that the system can tolerate. This is simply one component less than the total components in a cut set. The number of such test states is equal to the total number of components in the cut set. This approach, however, does not consider the combinations of the elements in one cut set with the elements of the other cut sets. The effects of these fault combinations are unknown if not tested. The maximal test sets do consider cross-combinations among several cut sets and so, in general, contain more elements than just "all but one component" from a cut set.

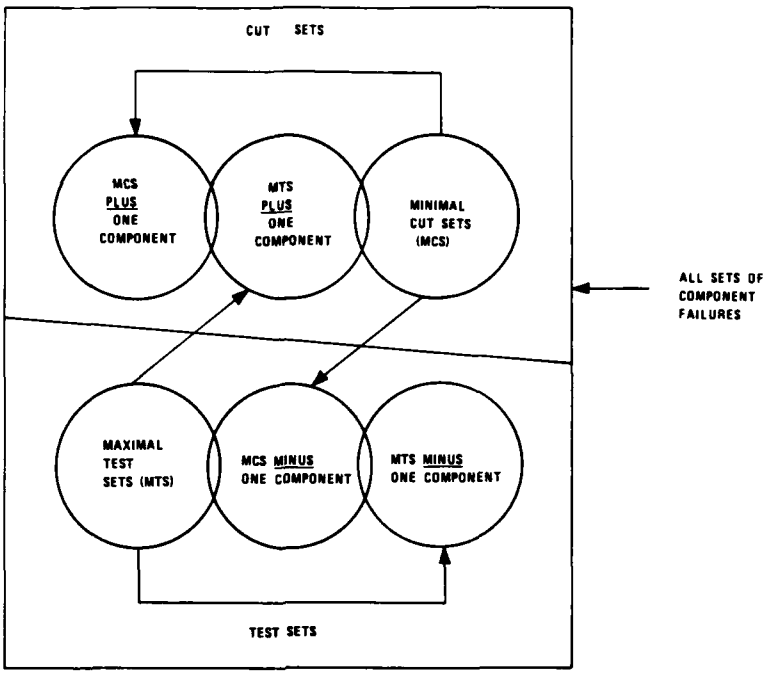


FIGURE 7 - **Any combination of component failures is either a cut set or a test set.**

These definitions allow one to state that a system works at least as often as its fault tree predicts if it works correctly for each combination of component failures in a test set. However, to test all the combinations in the test sets is still impractical for large, complex systems. One would like to use the maximal test sets for testing purposes in a way analogous to using minimal cut sets for reliability analysis. This requires the following assumption:

If a system fails under a given set of component faults, then it will fail under the given set of component faults plus any additional component faults.

This assumption is necessary to avoid having to test all combinations of faults in the test sets. With this assumption, it suffices to demonstrate the capablity of the system to operate under the combination of faults in each maximal test set.

These ideas will be made more concrete by means of the following example.

3.3 Test Set Example -- The example presented in this section is taken from an architecture study performed by a fly-by-wire flight control system for the Lockheed S-3A aircraft (ref. 11). As part of this study, Lockheed performed a fault-tree analyses to examine various architectures and estimate the effects of equipment failures rates on total system reliability. The top event of the fault tree corresponded to "loss of flight control". A total of 62 components were evaluated as part of the fault tree analysis. Various failure rates were varied parameterically. A list of the fault tree elements and typical failure rate data is given in Table 6. Computer results provide all minimal cuts sets and their associated failure probability. To improve overall system reliability, attention should be focused on components in the top-ranked sets. Table 7 lists the 25 most likely cut sets for one of the configurations. This date will be used to illustrate the development of test sets.

Maximal test sets--the largest sets not containing a cut set will be found for this example. Cut set numbers 1, 2, 6, 11, 12, and 13 are independent minimal cut sets in that each of these sets has no elements in common with any other minimal cut set. Cut set number 11 is typical. Three elements are contained in this cut set--the three redundant channels of the aileron secondary actuator (RS1, RS2, RS3). If all three servos fail, the roll channel fails, resulting in loss of aircraft control. The maximum number of aileron secondary actuator channel faults that the system can tolerate two. Since aileron secondary actuator channels do not appear in any other minimal cut sets, any maximal test set must contain exactly two aileron secondary actuator channel faults. Similarly, each maximal test set must contain exactly one element from cut set number 1 (since it has two elements) and two elements from cut set numbers 2, 6, 12, and 13. These cases are listed in table 8.

The remaining minimal cut sets may be divided into two groups. One group is cut set numbers 4, 5, 6 and the other group is numbers 7-10, 14-25. The two groups have no elements in common. For both groups, their contribution to maximal test sets is determined readily by inspection. The first group is illustrated in the Venn diagram of Figure 8. Since each of these cut sets has the elements IDG1 and IDG2 in common, the maximal test sets fall into one of three cases:

(1) All the elements except IDG1 are included
(2) All the elements except IDG2 are included
(3) Only IDG1 and IDG2 are included

The second group, consisting of cut sets 7-10 and 14-25, has a symmetry which can be exploited. Cut sets 7-10 are all combinations of three faults out of four air data computers. Cut sets 14-25 are all combinations of one sensor bus failure and two faults out of the other three air data computes. (As an aside, two sensor bus failures and one air data computer fault is also a minimal cut set, but is less likely than the 5 shown in table 9.) The maximal test sets must contain failures of two air data computers and the two corresponding sensor buses. There are six such combinations. All the maximal test sets equals the product of the number of elements in each independent minimal cut set and the number of combinations from each dependent group of minimal cut ses. For the example problem, this is

$$2 \times (3)^5 \times 3 \times 6 = 8748$$

In contrast, the total number of combinations of failures for the 30 components in table 9 is

$$2^{30} \approx 10^9$$

Testing only the maximal test sets results in tremendous savings. Moreover, it is sufficient to guarantee the performance of the system under the assumptions stated above.

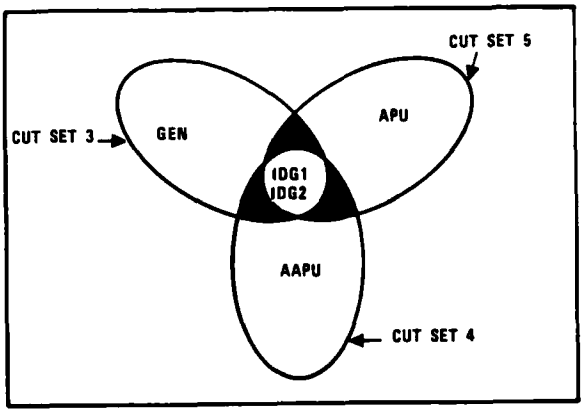


FIGURE 8 **- There are only three groupings of the elements of cut sets 3, 4, 5 into maximal test sets.**

Considering only maximal test sets offers large savings in the number of required test states. Additional savings are realized if the system can be partitioned into groups on the basis of failure modes and effects analysis, or some other such analysis. The danger here lies in omitting some failure mode from the fault-tree analysis and then partitioning the system on the same basis. Still further reductions in the amount of testing may be obtained by calculating the probability of occurrence for each maximal test set. If this probability is sufficiently low, then the set can be eliminated as a test state. This elimination amounts to saying that the failure combination is so remote that the system is allowed to fail in response to that combination.

It appears feasible to construct a computer program to generate all the maximal cut sets. Additional research and development efforts are needed to establish this feasibility.

TABLE 6 - ADFBW FAULT TREE PRIMITIVES

No.	Computer Symbol	Failure Rate (10^{-6}/hour)	Description
1	AAPU	438	Accumulator for APU starter
2	AC1	91	Air data computer, channel 1
3	AC2	91	Air data computer, channel 2
4	AC3	91	Air data computer, channel 3
5	AC4	91	Air data computer, channel 4
6	APU	480	Auxiliary power unit
7	AX1	30	Longitudinal accelerometer, channel 1
8	AX2	30	Longitudinal accelerometer, channel 2
9	AX3	30	Longitudinal accelerometer, channel 3
10	AX4	30	Longitudinal accelerometer, channel 4
11	AY1	30	Lateral accelerometer, channel 1
12	AY2	30	Lateral accelerometer, channel 2
13	AY3	30	Lateral accelerometer, channel 3
14	AY4	30	Lateral accelerometer, channel 4
15	AZ1	30	Vertical accelerometer, channel 1
16	AZ2	30	Vertical accelerometer, channel 2
17	AZ3	30	Vertical accelerometer, channel 3
18	AZ4	30	Vertical accelerometer, channel 4
19	BPMP	38.7	Backup pump for hydraulic system no. 1
20	BT1	348.9	Battery no. 1
21	BT2	348.9	Battery no. 2
22	CC1	250.0	FBW computer, channel 1
23	CC2	250.0	FBW computer, channel 2
24	CC3	250.0	FBW computer, channel 3
25	CT1	40.0	Copilot stick transducer, channel 1
26	CT2	40.0	Copilot stick transducer, channel 2
27	CT3	40.0	Copilot stick transducer, channel 3
28	ENG1	247.0	Engine no. 1
29	ENG2	247.0	Egine no. 2
30	GTEN	114.0	APU electrical generator
31	IDG1	5000.0	Integrated drive generator no. 1
32	IDG2	5000.0	Integrated drive generator no. 2
33	PG1	30.0	Pitch rate gyro, channel 1
34	PG2	30.0	Pitch rate gyro, channel 2
35	PG3	30.0	Pitch rate gyro, channel 3
36	PG4	30.0	Pitch rate gyro, channel 4
37	PMP1	209.0	Engine-driven hydraulic pump no. 1
38	PMP2	209.0	Engine-driven hydraulic pump no. 2
39	PS1	90.0	Elevator secondary actuator, channel 1
40	PS2	90.0	Elevator secondary actuator, channel 2
41	PS3	90.0	Elevator secondary actuator, channel 3
42	PT1	40.0	Pilot stick transducer, channel 1
43	PT2	40.0	Pilot stick transducer, channel 2
44	PT3	40.0	Pilot stick transducer, channel 3
45	RG1	30.0	Roll rate gyro, channel 1
46	RG2	30.0	Roll rate gyro, channel 2
47	RG3	30.0	Roll rate gyro, channel 3
48	RG4	30.0	Roll rate gyro, channel 4
49	RS1	90.0	Aileron secondary actuator, channel 1
50	RS2	90.0	Aileron secondary actuator, channel 2
51	RS3	90.0	Aileron secondary actuator, channel 3
52	SB1	10.0	Sensor bus, channel 1
53	SB2	10.0	Sensor bus, channel 2
54	SB3	10.0	Sensor bus, channel 3
55	SB4	10.0	Sensor bus, channel 4
56	YG1	30.0	Yaw rate gyro, channel 1
57	YG2	30.0	Yaw rate gyro, channel 2
58	YG3	30.0	Yaw rate gyro, channel 3
59	YG4	30.0	Yaw rate gyro, channel 4
60	YS1	90.0	Rudder secondary actuator, channel 1
61	YS2	90.0	Rudder secondary actuator, channel 2
62	YS3	90.0	Rudder secondary actuator, channel 3

TABLE 7 - MINIMAL CUT SET DATA IN DESCENDING ORDER OF PROBABILITY

Cut Set Number	Maximum Failure Probability	Components Contained in Set	Description of Mnemonic
1	0.99989848E-08	ENG1 ENG2	Engines
2	0.15619062E-10	CC1 CC2 CC3	Computer channels
3	0.99984777E-12	GEN IDG1 IDG2	APU generator, integrated drive generators
4	0.99984777E-12	AAPU IDG1 IDG2	APU accumulator, integrated drive generators
5	0.99984777E-12	APU IDG1 IDG2	Auxiliary power unit, integrated drive generators
6	0.99984777E-12	BPMP PMP2 PMP1	Backup pump, pumps
7	0.75346635E-12	AC1 AC2 AC3	Air data computers
8	0.75346635E-12	AC4 AC2 AC3	
9	0.75346635E-12	AC4 AC1 AC2	
10	0.75346635E-12	AC4 AC1 AC3	
11	0.72889996E-12	RS1 RS2 RS3	Aileron secondary actuators
12	0.72889996E-12	PS1 PS2 PS3	Elevator secondary actuators
13	0.72889996E-12	YS1 YS2 YS3	Rudder secondary actuators
14	0.82801821E-13	AC1 AC2 SB3	Air data computers, sensor bus
15	0.82801821E-13	AC4 AC2 SB3	
16	0.82801821E-13	AC4 AC1 SB3	
17	0.82801821E-13	AC4 AC1 SB2	
18	0.82801821E-13	SB4 AC1 AC2	
19	0.82801821E-13	AC1 SB2 AC3	
20	0.82801821E-13	AC4 SB2 AC3	
21	0.82801821E-13	AC4 SB1 AC3	
22	0.82801821E-13	AC4 SB1 AC2	
23	0.82801821E-13	SB1 AC2 AC3	
24	0.82801821E-13	SB4 AC2 AC3	
25	0.82801821E-13	SB4 AC1 AC3	

TABLE 8 - MAXIMAL TEST SETS FOR THE EXAMPLE ARE CONSTRUCTED AS THE UNION OF ONE SUBSET FROM EACH OF THE EIGHT INDEPENDENT GROUPS

Cut Set Number	Intersection of the Cut Sets with Maximal Test Sets (failed components)
1	{ENG1}, {ENG2}
2	{CC1,CC2}, {CC1,CC3}, {CC2,CC3}
6	{BPMP,PMP1}, {BPMP,PMP2}, {PMP1,PMP2}
11	{RS1,RS2}, {RS1,RS3} {RS2,RS3}
12	{PS1,PS2}, {PS1,PS3}, {PS2,PS3}
13	{YS1,YS2}, {YS1,YS3}, {YS2,YS3}
4,5,6	{GEN,AAPU,APU,IDG1}, {GEN,AAPU,APU,IDG2}, {IDG1,IDG2}
7-10,14-25	{AC1,AC2,SB1,SB2}, {AC1,AC3,SB1,SB3}
	{AC1,AC4,SB1,SB4}, {AC2,AC3,SB2,SB3}
	{AC2,AC4,SB2,SB4}, {AC3,AC4,SB3,SB4}

REFERENCES

1. "Validation Methods for Fault-Tolerant Avionics and Control Systems", NASA Working Group Meeting No. 1, 12-14 March, 1979.

2. Roubine, O.; and Robinson, L.: Special Reference Manual. Third Edition. Stanford Research Institute, Menlo Park, California, Report No. CSG-45, 1977.

3. Military Standard: Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs. MIL-STD-483 (USAF). Notice 2, March 21, 1979.

4. Heinger, K.L.: Specifying Software Requirements for Complex Systems: New Techniques and Their Applications. Proc., Specification of Reliable Software. Cambridge, Massachusetts, April 3-5, 1979, pp. 1-14.

5. Peterson, J.L.: Petri Nets. ACM Computing Surveys, vol. 9, no. 3, September 1977, pp. 223-252.

6. Jack, L.A.; Heimerdinger W.L.; and Johnson, M.D.: Theory of Fault Tolerance. Honeywell Systems and Research Center, Minneapolis, Minnesota, 1974-5 Annual Report, September 1975.

7. Han, Y.W.; and Heimerdinger, W.L.: Theory of Fault Tolerance. Honeywell Systems and Research Center, 1977 Final Report, 77SRC82, Minneapolis, Minesota, December 1977.

8. Heimerdinger, W.L.; and Fant, K.M.: A Fault Tolerant Assessment of DAIS. Air Force Avionics Laboratory, Wright-Patterson Air Force Base, Ohio, AFAL-TR-79-1007, March 1979.

9. E.R. Rang, "The Use of Finite-State Machines for Describing and Validating Flight Control Systems", Proceedings of NAECON '80, Vol. 1, pp. 347-353, Dayton, Ohio, May, 1980.

10. Pease, M.; Shostak, R.; and Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM, vol. 27, no. 2, April 1980, pp. 228-234.

11. Hartmann, G.L. et al "Advanced Flight Control System Study" Final Report Contract NASA-2876, Report No. 163117, February 1982 (available through National Technical Information Service Springfield, VA 22151)

AGARD Lecture Series No. 143

Fault Tolerance Software/Hardware Architecture

For Flight Critical Function

This Bibliography with Abstracts has been prepared to support AGARD Lecture Series No. 143 by the Scientific and Technical Information Branch of the U.S. National Aeronautics and Space Administration, Washington, D.C., in consultation with the Lecture Series director, Mr. Gary L. Hartmann, Honeywell, Minneapolis, Minnesota.

UTTL: Digital avionics - The best is yet to come
AUTH: A/SPITZER, C. R. PAA: A/(NASA, Langley Research Center, Hampton, VA) CORP: National Aeronautics and Space Administration, Langley Research Center, Hampton, Va. IEEE Transactions on Aerospace and Electronic Systems (ISSN 0018-9251), vol. AES-20, July 1984, p. 486-492.
ABS: The history and background of digital avionics and the possibilities for the future are reviewed. There are payoffs in many areas from digital avionics; however, the ultimate benefits are increased mission effectiveness and lower costs. Two major U.S. Air Force avionics programs designed to increase mission effectiveness are reviewed. Major barriers to the expanded use of digital avionics in civil transports as a means to lower operating costs are examined. The paper also examines lightning effects, architectures, optical components, displays, and voice-interactive control, which are current research areas that promise to yield significant advances for digital avionics systems. 84/07/00 84A47693

UTTL: A310/A300-600 electronic flight instrument system
AUTH: A/POTOCKI, P. PAA: A/(Airbus Industrie, Toulouse, France) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineers, 1983, p. 22.3.1-22.3.7.
ABS: The principal design and performance characteristics of the Electronic Flight Instrument System (EFIS) installed on the large transports A310 and A300-600 are discussed. The EFIS combines the display flexibility of the cathode ray tube and the computing power of the microprocessor to provide the crew with an easily assimilated display of the aircraft situation in four-dimensional space. The system is particularly easy to troubleshoot down to the Line Replaceable Unit level, minimizing the effect on schedule reliability. Another advanced feature is the use of common components that enable aircraft dispatch for a limited number of flight legs with a failed component. The EFIS Fault Isolation and Detection System includes a nonvolatile memory for intermittent faults as well as built-in test equipment that will identify a failed unit or input. 83/00/00 84A26800

UTTL: Broad range of programmable fault tolerance in transition machine computer architectures
AUTH: A/VAUGHAN, R. F. PAA: A/(Boeing Aerospace Co., Seattle, WA) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineers, 1983, p. 21.2.1-21.2.6.
ABS: Transition machines are examined in terms of reliability considerations, configuration variations, real-time recovery procedures, and programming to handle faults. Recent research has defined classes of redundant component organization endowed with superior reliabilities; a broad range of source programmable fault tolerance has been defined for these organizations, comprising a spectrum from simple fault detection through triplicated processing pipes, each with its own failure correction capability. HOL commands are used to effect a user-specified degree of fault tolerance; these programmed designations can be based on the criticality of the individual computations to the overall avionics mission success. It is concluded that transition machine computer architectures form an ideal basis for fault-tolerant avionic computing systems. 83/00/00 84A26796

UTTL: A review and application of analytical models in fault tolerant avionics validation
AUTH: A/HITT, E. F.; B/ELDREDGE, D. PAA: A/(Battelle Columbus Laboratories, Columbus, OH); B/(FAA, Technical Center, Atlantic City, NJ) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineers, 1983, p. 16.4.1-16.4.8.
ABS: The paper examines the general capabilities and limitations of (1) five reliability models developed specifically for the evaluation of fault-tolerant avionics systems (ARIES, CARE II and III, CARSRA, and CAST); (2) fault trees; and (3) failure modes and effects analysis. A description is given of the complementary use of these three classes of models. It is noted that all the methods require similar input data which is generally inaccurate until the system development is nearly complete; methods for dealing with inadequate data are presented. It is concluded that analytical models and methods play a major role in all phases of the development of an avionics system, including the system validation and certification; no single method is adequate by itself or for all phases. 83/00/00 84A26774

UTTL: Fault, detection, isolation, and recovery techniques for fault tolerant digital avionics
AUTH: A/HITT, E. F.; B/ELDREDGE, D. PAA: A/(Battelle Columbus Laboratories, Columbus, OH); B/(FAA, Technical Center, Atlantic City, NJ) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineers, 1983, p. 16.1.1-16.1.8.
ABS: Fault tolerant design technologies for digital avionics system are described in this paper. The techniques include both hardware and software methods used for detecting faults at three levels. These levels should be implemented to assure (1) the correct operation of each processing unit, (2) valid communication of data between digital subsystems, and (3) data validity, prior to use in subsequent computation and after conversion of digital data. Once a fault is detected, system recovery must take place to assure the continued performance of the function(s) affected by the fault. The methods used to control the system recovery techniques are dependent upon the system's ability to isolate the detected fault to the lowest possible level. The system recovery techniques are also dependent upon the system architecture. Fault isolation and system recovery techniques require knowledge of the system status vector and its history in sophisticated systems. 83/00/00 84A26771

UTTL: Fault tolerant flight control avionics integration using MIL-STD-1553B
AUTH: A/MCSHARRY, M. E. PAA: A/(Boeing Military Airplane Co., Seattle, WA) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineers, 1983, p. 11.1.1-11.1.8.
ABS: While the design of integrated systems using distributed processing, hierarchical architectures, and data bases, provides the greatest independence and immunity from fault propagation, compromises that tend to more tightly couple integrated system components may be needed in order to satisfy performance requirements. Attention is presently given to the MIL-STD-1553B integrated system data bus, which is marginally capable of satisfying the data transfer requirements for both flight control and mission avionics and whose mission avionics functions must be implemented with a higher level of redundancy if the mission functions affect flight safety. Redundancy can be attained through hardware replication as well as analysis. 83/00/00 84A26744

UTTL: Conceptual design of a digital avionics suite for LHX
AUTH: A/PRUYN, R. R. PAA: A/(Boeing Vertol Co., Philadelphia, PA) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineering, 1983, p. 3.3.1-3.3.5.
ABS: Marner and Pruyn (1983) have discussed the design of the Army's new light LHX helicopter. The present investigation represents a continuation of this discussion, taking into account the requirements of the LHX avionics and suggestions regarding the partition of the system. It is pointed out that the LHX will require development of a highly capable digital avionics suite. Attention is given to the reliability of LHX avionics, aspects of design philosophy, alternatives to architectural partitioning, the functional arrangement of avionics, the concept of three-partition architecture, considerations for fault-tolerant processors, and the modular configuration of internally redundant processor. 83/00/00 84A26706

UTTL: The missing link for advanced avionics systems executives
AUTH: A/LEEPER, K. R. PAA: A/(Boeing Military Airplane Co., Advanced Airplane Branch, Seattle, WA) IN: Digital Avionics Systems Conference, 5th, Seattle, WA, October 31-November 3, 1983, Proceedings (A84-26701 11-06). New York, Institute of Electrical and Electronics Engineers, 1983, p. 2.6.1-2.6.6.
ABS: An avionics system executive was developed with the aid of the Digital Avionics Information System (DAIS) program. This executive was coded mostly in high-order language with hardware interfaces in machine code. However, it was found that the DAIS executive was more complex than necessary for many applications. It was, therefore, decided to eliminate asynchronous operations from the executive. As a result of this decision the Single Processor Synchronous Executive (SPSE) was obtained. Developments with respect to a further evolution of standards continued, however, and revisions appeared which were not included in the DAIS evolution. The present investigation is concerned with the efforts of an American aerospace company to update the SPSE to MIL-STD-1750A and MIL-STD-1589B. It is pointed out that the 1750A SPSE represents the missing link in the evolution of the avionics executive of yesterday to the advanced executive of tomorrow. 83/00/00 84A26704

UTTL: Implementing microprocessor technology in aircraft electrical power generating system control

AUTH: A/LORENZ, S.; B/MEHL, B.; C/RUFFNER, G. PAA: C/(Sundstrand Corp., Advanced Technology Group, Rockford, IL) IN: NAECON 1983; Proceedings of the National Aerospace and Electronics Conference, Dayton, OH, May 17-19, 1983. Volume 1 (A84-16526 05-01). New York, Institute of Electrical and Electronics Engineers, 1983, p. 123-134.

ABS: Today's advanced aircraft electrical power generating systems rely on microprocessor technology for the implementation of most control, protection, and built-in test functions. Microprocessors offer distinct advantages over discrete logic devices in system design and performance. The first half of this paper highlights these advantages by illustrating the design implementation process used in current systems. The second half of the paper expands on these advantages. By adapting more advanced microprocessor systems in the next generation of aircraft electric systems, additional functions can be implemented. Microprocessor control of generator paralleling and voltage regulation coupled with more effective built-in test capabilities will result in significant improvements in system performance. 83/00/00 84A16536

UTTL: Avionic architectures for fly-by-wire aircraft

AUTH: A/PAPADOPOULOS, G. M. PAA: A/(Honeywell Systems and Research Center, Minneapolis, MN) IN: NTC '82; National Telesystems Conference, Galveston, TX, November 7-10, 1982, Conference Record (A84-15623 04-32). New York, Institute of Electrical and Electronics Engineers, Inc., 1982, p. 85.5.1-85.5.3.

ABS: Advances in flight control, airframe, engine, communication, and instrumentation have led to increases in complexity. Modern designs are characterized by isolated, independently designed subsystems. It has been found that reliability is degraded as complexity is increased and that the full performance potential of the system is not realized. For an achievement of higher performance, a system should be so structured as to allow the free flow of information from one subsystem or function to another without endangering the reliable operation of either. Attention is given to safety of flight reliability, flight control architectures, the Multiple Microprocessor Flight Control System Study (MMFCS), and the generalization of MMFCS concepts to the rest of the avionics system. 82/00/00 84A15652

UTTL: Integrated digital avionic systems - Promise and threats

AUTH: A/ZEMPOLICH, B. A. PAA: A/(U.S. Naval Air Systems Command, Washington, DC) Astronautics and Aeronautics (ISSN 0004-6213), vol. 21, Oct. 1983, p. 46-53.

ABS: The progress being made in effective systems design implementation for digital equipment for aircraft avionics sytems is assayed. The history of digital systems integration in avionics hardware is traced from use of 16-transistor chips to emerging 100,000 gate chips, and attention is given to architectural considerations for future hardware. Design considerations include top-down or bottom-up architecture, distributed microprocessor and computer resources, integrated components or data fusion, etc. Systems decomposition practices in design permit separate design of flight safety systems, redundancy, fault tolerance, and identifying components that feature different technologies. Present flight control systems sport a MBTF of 1,000,000 hr when separate controls are installed for each flight system. 83/10/00 83A48890

UTTL: F-14 aircraft and propulsion control integration evaluation

AUTH: A/DAVIES, W. J.; B/HOELZER, C. A.; C/VIZZINI, R. W. PAA: A/(United Technologies Corp., Government Products Div., West Palm Beach, FL); B/(Grumman Aerospace Corp., Bethpage, NY); C/(U.S. Naval Air Propulsion Test Center, Trenton, NJ) American Society of Mechanical Engineers, International Gas Turbine Conference and Exhibit, 28th, Phoenix, AZ, Mar. 27-31, 1983. 10 p.

ABS: An integration evaluation is presented for a fault-tolerant, full authority digital electronic control (FADEC) in an F-14 aircraft, and the benefits of the FADEC/F-14 integrated system are discussed. The control of the advanced fuel management system incorporated into the overall system is addressed, as are the flight propulsion control system integration, the inlet and engine airflow matching, stall detection and recovery, and improved cruise fuel consumption.

RPT#: ASME PAPER 83-GT-234 83/03/00 83A48029

UTTL: A method of designing fault tolerant software

AUTH: A/SHEPHERD, J. PAA: A/(Cranfield Institute of Technology, Cranfield, Beds., England) IN: Certification of avionic systems; Proceedings of the Symposium, London, England, April 27, 1982 (A83-45843 22-01). London, Royal Aeronautical Society, 1982, 7 p.

ABS: A method for the design of fault-tolerant avionics

software is presented which incorporates the temporal separation of software channels and the derivation of an alternative version of a primary version of a program. Temporal separation ensures that no software fault affects all data channels simultaneously, as is presently illustrated for the case of a triplex system. Alternative program versions encompass arithmetic, logical, decision, and input/output operations, including interrupts. Attention is given to the estimation of the probability of failure in a fault-tolerant system incorporating these software design principles. 82/00/00 83A45847

UTTL: Flight clearance of the Jaguar-fly-by-wire aircraft. II
AUTH: A/SMITH, R. B. PAA: A/(Marconi Avionics, Ltd., Rochester, Kent, England) IN: Certification of avionic systems; Proceedings of the Symposium, London, England, April 27, 1982 (A83-45843 22-01). London, Royal Aeronautical Society, 1982, 10 p. Research supported by the Ministry of Defence (Procurement Executive) and British Aerospace PLC.
ABS: The Integrated Flight Control System of the Jaguar fly-by-wire aircraft has as its primary integrity requirement that it possess a dual fault failure survival capability. This requirement is addressed by means of a quadruplex configuration. A flight test assessment of this design has shown it to achieve a safety-critical risk level of 2.64 x 10 to the -7th/flight hour. It is noted that the failure survival characteristics of the system are primarily dependent on its architecture, together with the performance of its voter/monitors and actuators. With respect to the development of high integrity software, the present appraisal has shown that the greatest single source of error is in the design definition phase. 82/00/00 83A45846

UTTL: Certification of avionic systems; Proceedings of the Symposium, London, England, April 27, 1982
Symposium sponsored by the Royal Aeronautical Society. London, Royal Aeronautical Society, 1982, 85 p.
ABS: Among the topics discussed are military perspectives in the certification of avionics systems, by contrast with those of civil aviation, military avionics system acceptance criteria developed for state-of-the-art digital and multisensor systems, the quadruplex digital fly-by-wire system of the Jaguar fighter, a method for the design of fault-tolerant avionics software, and certification procedures for the digital avionics systems of such commercial aircraft as the 757, 767, and A310 airliners. Also considered are system certification methods for uniquely helicopter control-related avionics, and the certification case history of the L 1011-500 airliner's maneuver load-controlling and elastic mode suppressing active control system. For individual items see A83-45844 to A83-45850 82/00/00 83A45843

UTTL: New results in fault latency modelling
AUTH: A/MCGOUGH, J. G.; B/SWERN, F. L.; C/BAVUSO, S. PAA: B/(Bendix Corp., Teterboro, NJ); C/(NASA, Langley Research Center, Langley, VA) CORP: Bendix Corp., Teterboro, N.J.; National Aeronautics and Space Administration. Langley Research Center, Hampton, Va. IN: Guidance and Control Conference, Gatlinburg, TN, August 15-17, 1983, Collection of Technical Papers (A83-41659 19-63). New York, American Institute of Aeronautics and Astronautics, 1983, p. 882-889.
ABS: Studies carried out by McGough and Swern (1981, 1983) are summarized. In these studies, an avionics processor was simulated and a series of fault injection experiments was carried out to determine the degree of fault latency in a redundant flight control system that employed comparison monitoring as the exclusive means of failure detection. A determination was also made of the fault coverage of a typical self-test program. The summary presented stresses that a self-test program should be designed to capitalize on the hardware mechanization of the processor. If this is not done, subtests tend to repeatedly exercise the same hardware components while neglecting to exercise a substantial proportion of the remainder. It is also pointed out that fault latency is relatively independent of both the length and instruction mix of a program. A significant difference is found in fault coverage assessed using pin-level and gate-level fault models
RPT#: AIAA PAPER 83-2303 83/00/00 83A41760

UTTL: Fault isolation methodology for the L-1011 digital avionic flight control system
AUTH: A/NOBLE, W. B. PAA: A/(Hughes Aircraft Co., Fullerton, CA) (Digital Avionics Systems Conference, 4th, St. Louis, MO, November 17-19, 1981, Collection of Technical Papers, p. 56-61.) Journal of Guidance, Control and Dynamics, vol. 6, Mar.-Apr. 1983, p. 72-76.
ABS: (Previously cited in issue 03, p. 331, Accession no. A82-13458) 83/04/00 83A24427

UTTL: A prototype parallel computer architecture for advanced avionics applications

AUTH: A/ANASTAS, M. S.; B/VAUGHAN, R. F. PAA: B/(Boeing Aerospace Corp., Seattle, WA) In: NAECON 1982; Proceedings of the National Aerospace and Electronics Conference, Dayton, OH, May 18-20, 1982. Volume 2. (A83-11083 01-01) New York, Institute of Electrical and Electronics Engineers, Inc., 1982, p. 614-621.

ABS: Performance data have been obtained which demonstrate the practicality of the transition machine architecture (a new type of parallel computer architecture) and verify the accuracy of an analytic model (Vaughan and Anastas, 1980) which predicts the performance of tightly coupled multiprocessor systems. These demonstrated features land credence to claims for effectively combining large numbers of microprocessors in tightly coupled configurations using transition machine concepts. The efficiency of the transition machine architecture and its amenability to conventional HOL programming provide low cost, easily programmed systems which can be used to obtain high throughput, fault tolerance, and modular expendability. 82/00/00 83A11153

UTTL: Radiation and chemistry in the stratosphere - Sensitivity to O2 absorption cross sections in the Herzberg continuum

AUTH: A/FROIDEVAUX, L.; B/YUNG, Y. L. PAA: B/(California Institute of Technology, Pasadena, CA) CORP: California Inst. of Tech., Pasadena. Geophysical Research Letters, vol. 9, Aug. 1982, p. 854-857. NASA-supported research.

ABS: It is suggested that the discrepancies between observed and modeled vertical profiles of such halocarbons as CFCl3, as well as the problem of simultaneously fitting N2O, CH4, CF2Cl2 and CFCl3 profiles with a single eddy diffusion model, are due to an overestimation of the molecular oxygen absorption cross sections in the 200-220 nm spectral region. The replacement of current O2 cross sections in this range with values that are in better agreement with results for the compounds cited leads to N2O, CF2Cl2 and CFCl3 concentration reductions of factors 0.70, 0.62 and 0.19, respectively. Profiles of CH4, H2 and CO remain unchanged, and the predicted concentration of HNO3 above 30 km is reduced by about 50% for yet another improved fit with observations. It is noted that the correction proposed produces a 30% ozone increase near the 20-25 km peak. 82/08/00 82A42237

UTTL: Phased-mission analysis for evaluating the effectiveness of aerospace computing systems

AUTH: A/PEDAR, A.; B/SARMA, V. V. S. PAA: A/(National Aeronautical Laboratory, Bangalore, India); B/(Indian Institute of Science, Bangalore, India) IEEE Transactions on Reliability, vol. R-30, Dec. 1981, p. 429-437.

ABS: A model of a distributed fault-tolerant avionic system of a transport aircraft is presented as an example of a phased-mission analysis for aerospace computing systems. The avionic system consists of a flight control, inertial navigation system, radio navigational aids, fuel management, and autolanding system computers. A demand profile for a flight is constructed for fuel efficient flight, constant failure rates for all computers, and each phase having an s-coherent structure. The probabilities of accomplishing desired levels are calculated. An evaluation is made of the bounds of effectiveness for the computing systems, with considerations of model development for a multistate system, an analysis of the system components, and the use of a hazard transform bound. An example is provided which includes a provision for the functions of a failed computer to be carried out by the remaining systems. 81/12/00 82A23350

UTTL: Fault secure avionic system development

AUTH: A/JENNINGS, R. PAA: A/(USAF, Wright Aeronautical Laboratories, Wright-Patterson AFB, OH) In: NAECON 1981; Proceedings of the National Aerospace and Electronics Conference, Dayton, OH, May 19-21, 1981. Volume 1. (A82-14676 04-01) New York, Institute of Electrical and Electronics Engineers, Inc., 1981, p. 284-293.

ABS: With the technological improvements that have been made in computer hardware, major limitations now have to do with programmability, integrity, and reliability. It is contended that these limitations can be largely solved at the computer and integrated circuit architecture level through an organizational concept called Fault Secure Avionic Computer (FSAC). The kernel of the FSAC consists of a programmable processor, of a type suitable for mass production, which has provisions for exploiting special purpose VLSI arithmetic and data management hardware to expedite execution of time critical tasks. 81/00/00 82A14714

UTTL: NAECON 1981; Proceedings of the National Aerospace and Electronics Conference, Dayton, OH, May 19-21, 1981. Volumes 1, 2 & 3 Conference sponsored by the Institute of Electrical and Electronics Engineers. New York, Institute of Electrical and Electronics Engineers, Inc., 1981. Vol. 1, 460 p.; vol. 2, 495 p.; vol. 3, 502 p. (For individual items see A82-14677 to A82-14843)

ABS: Topics of aerospace electronics such as the ADA programming language, inertial systems, microcomputer applications, survivability, and the all-electric aircraft were discussed. Papers were presented on laser gyros and advanced navigation systems, as well as advanced architecture, communications, and radar equipment, software, and avionics and armament planning. Failures in high voltage tubes were considered, and attention was given to signal processing techniques, integrated aircraft controls, fire control, software support tools, cost estimates for software, and medical technology. Emphases were placed on Kalman filter, an electronic terrain map, FM compatibility, aerospace power systems, air traffic control, environmental stress measurements, fault isolation, and multivariable flight control design. 81/00/00 82A14676

UTTL: Methodology for measurement of fault latency in a digital avionic miniprocessor

AUTH: A/MCGOUGH, J. G.; B/SWERN, F.; C/BAVUSO, S. J.

PAA: B/(Bendix Corp., Flight Systems Div., Teterboro, NJ); C/(NASA, Langley Research Center, Hampton, VA)

CORP: Bendix Corp., Teterboro, N.J.; National Aeronautics and Space Administration, Langley Research Center, Hampton, Va. In: Digital Avionics Systems Conference, 4th, St. Louis, MO, November 17-19, 1981, Collection of Technical Papers. (A82-13451 03-04) New York, American Institute of Aeronautics and Astronautics, 1981, p. 301-314.

ABS: Investigations regarding the synthesis of a reliability assessment capability for fault-tolerant computer-based systems have been conducted for several years. In 1978 a pilot study was conducted to test the feasibility of measuring detection coverage and investigating the dynamics of fault propagation in a digital computer. A description is presented of an investigation concerned with the applicability of previous results to a real avionics processor. The obtained results show that emulation is a practicable approach to failure modes and effects analysis of a digital processor. The run time of the emulated processor on a PDP-10 host computer is only 20,000 to 25,000 times slower than the actual processor. As a consequence large numbers of faults can be studied at relatively little cost and in a timely manner.

RPT#: AIAA 81-2282 81/00/00 82A13493

UTTL: Computer-in-control selection logic for a triplex digital flight control system

AUTH: A/FERRELL, P. J. PAA: A/(Boeing Military Airplane Co., Seattle, WA) In: Digital Avionics Systems Conference, 4th, St. Louis, MO, November 17-19, 1981, Collection of Technical Papers. (A82-13451 03-04) New York, American Institute of Aeronautics and Astronautics, 1981, p. 97-101.

ABS: The computer-in-control logic (CICL) unit is an independent monitor that ensures proper selection of one of two flight control computers to the simplex level with minimum switching. The CICL maximizes fault tolerance without being a source of a single-point failure; its implementation is simple and reliable, and its algorithm is easy to change. This paper presents a functional description of the CICL and examines potential architectural considerations.

RPT#: AIAA 81-2236 81/00/00 82A13465

UTTL: Fault isolation methodology for the L-1011 digital avionic flight control system

AUTH: A/NOBLE, W. B. PAA: A/(Rockwell International Corp., Collins Air Transport Div., Van Nuys, CA) In: Digital Avionics Systems Conference, 4th, St. Louis, MO, November 17-19, 1981, Collection of Technical Papers. (A82-13451 03-04) New York, American Institute of Aeronautics and Astronautics, 1981, p. 56-61.

ABS: The operation of and rationale for the fault isolation/data display system for the L-1011 digital avionic flight control system are described. These systems are currently in service and are, despite the usual and expected introductory problems, providing substantial maintenance benefit. The English-language format chosen for the display as well as the storing of the squawk (or event) which accompanied each fault, have been instrumental in reducing maintenance confusion and false LRU removals. Present status mode is being used to solve a number of intricate maintenance problems in minutes, which previously required hours of troubleshooting time.

RPT#: AIAA 81-2223 81/00/00 82A13458

UTTL: Digital Avionics Systems Conference, 4th, St. Louis, MO, November 17-19, 1981, Collection of Technical Papers Conference sponsored by the American Institute of Aeronautics and Astronautics and Institute of Electrical and Electronics Engineers. New York, American Institute of Aeronautics and

Astronautics, 1981. 645 p. (For individual items see A82-13452 to A82-13534)

ABS: Digital avionics are discussed in terms of a system integration concept, fault isolation methodology, system effectiveness, advanced designs, sneak software analysis, and the pilot's role in an automated flight deck. Specific applications for the L-1011 flight control system, for hardware/software integration on the Shuttle, for one man operation of the F/A-18 Hornet, with voice command control, and for advanced weapons systems were considered. Papers were also presented on individual components of digital avionics systems such as the MIL-STD-1750 chip set, standardization and semiconductors, fiber optics, connectors for data buses, large screen CRT touch panels, an electronic terrain map, and flat panels for future military aircraft. 81/00/00 82A13451

UTTL: The RLU - An advanced remote terminal for avionic systems

AUTH: A/TAVORA, C. J.; B/GLOVER, J. R., JR. PAA: B/(Houston, University, Houston, Tex.) In: NAECON 1980; Proceedings of the National Aerospace and Electronics Conference, Dayton, Ohio, May 20-22, 1980. Volume 3. (A81-30226 12-04) New York, Institute of Electrical and Electronics Engineers, Inc., 1980, p. 1271-1278.

ABS: This paper describes the design of the Remote Link Unit (RLU) at a functional level. The RLU is an intelligent remote terminal which is compatible with hierarchical distributed processing. It has interface modules capable of processing subsystem data on-the-fly and supporting a universal signal interface which can be configured under software control to input or output analog and digital signals of varied types. Electronic nameplates containing identification, interface requirements and signal processing programs are attached to avionics subsystems to support the RLU operational features. The RLU stand-alone processing capabilities facilitate maintenance and provide fault tolerance. The RLU is being evaluated by the Air Force for use in the Digital Avionics Information System (DAIS). 80/00/00 81A30358

UTTL: An advanced multiprocessor architecture - Skewed Processing

AUTH: A/BERLIN, E. P., JR. PAA: A/(Grumman Aerospace Corp., Bethpage, N.Y.) In: NAECON 1980; Proceedings of the National Aerospace and Electronics Conference, Dayton, Ohio, May 20-22, 1980. Volume 2. (A81-30226 12-04) New York, Institute of Electrical and Electronics Engineers, Inc., 1980, p. 862-866.

ABS: This paper describes a new computer architecture which combines multiple processors to achieve a throughput proportional to the number of processors. The system, which is called Skewed Processing, has distributed control functions, and lends itself to implementation of a fault tolerant system. If any processor fails, it may be 'removed' from the circuit and the remaining processors will distribute the task equally among themselves. The skewed architecture handles task partitioning transparently, and significant software development cost savings can be realized. A prototype skewed processor system is currently being built which implements a moving target indicator filter and demonstrates fault tolerance. 80/00/00 81A30321

UTTL: Fault tolerance applications to future military system avionics

AUTH: A/HARRIS, R. L.; B/JONES, E. E. PAA: B/(USAF, Avionics Laboratory, Wright-Patterson AFB, Ohio) In: NAECON 1980; Proceedings of the National Aerospace and Electronics Conference, Dayton, Ohio, May 20-22, 1980. Volume 1. (A81-30226 12-04) New York, Institute of Electrical and Electronics Engineers, Inc., 1980, p. 412-419.

ABS: Fault tolerance is the capability of a computer system to render a predesignated level of service after the occurrence of one or more hardware or software malfunctions. Fault tolerance operations include fault detection, fault isolation, and fault nullification. The inclusion of fault tolerance concepts in new avionic system designs has been implemented as Air Force policy. A review of current fault tolerance literature applicable to avionics system design is summarized, with an emphasis on multiplexed digital structures. Sample exploratory development programs that could affect the design of future avionic systems are presented. Fault tolerant avionics will increase aircraft availability if solutions to its implementation problems, including maintenance engineering, cost control, development of integrated systems, and standardization can be solved. 80/00/00 81A30273

UTTL: Using microprocessors in fault monitoring of aircraft electronics

AUTH: A/MAYBANKS, A. PAA: A/(Ultra Electronic Controls, Ltd., London, England) American Society of Mechanical Engineers, Gas Turbine Conference and Products Show, Houston, Tex., Mar. 9-12, 1981, 5 p.

ABS: The Ultra Electronic Controls Fault Identification Module, as used in the Electronic Engine Control Unit

(ECU) for the Olympus 593 engines of the Concorde Supersonic Transport Aircraft, is discussed. This is based on a CMOS microprocessor for low power consumption and enables the module to be applied to existing units without redesign of power supplies. The module examines the outputs of existing fault monitoring circuits and compares these with software-defined reference levels. It then determines, from this and other signals taken from the ECU safety consolidation circuits, the engine control subsystem which is at fault. This module has been in service for close to one year now and the impact on rapid and accurate fault diagnosis, elimination of premature ECU removals and thus reduction of cost ownership of the ECU is discussed.

RPT#: ASME PAPER 81-GT-138 81/03/00 81A30040

UTTL: SIFT - Multiprocessor architecture for Software Implemented Fault Tolerance flight control and avionics computers

AUTH: A/FORMAN, P.; B/MOSES, K. PAA: B/(Bendix Corp., Flight Systems Div., Teterboro, N.J.) CORP: Bendix Corp., Teterboro, N.J. In: Challenge of the '80s; Proceedings of the Third Digital Avionics Systems Conference, Fort Worth, Tex., November 6-8, 1979. (A80-32417 12-06) New York, Institute of Electrical and Electronics Engineers, Inc., 1979, p. 325-329.

ABS: A brief description of a SIFT (Software Implemented Fault Tolerance) Flight Control Computer with emphasis on implementation is presented. A multiprocessor system that relies on software-implemented fault detection and reconfiguration algorithms is described. A high level reliability and fault tolerance is achieved by the replication of computing tasks among processing units. 79/00/00 80A32464

UTTL: A state-of-the-art fault-tolerant computer

AUTH: A/FERNANDEZ, M. PAA: A/(Litton Industries, Guidance and Control Systems Div., Woodland Hills, Calif.) In: Challenge of the '80s; Proceedings of the Third Digital Avionics Systems Conference, Fort Worth, Tex., November 6-8, 1979. (A80-32417 12-06) New York, Institute of Electrical and Electronics Engineers, Inc., 1979, p. 319-324.

ABS: The study deals with an approach which provides a low-cost/low-risk, fault-tolerant computer that can experience hardware failures and keep computing correctly via built-in, autonomous, self-healing techniques for long periods of time and with high probability of mission success. It is shown that a middle-of-the-road approach (using proven, efficient fault tolerance techniques and proven, state-of-the-art microelectronics) gives a viable solution in that it optimizes cost/risk tradeoffs. 79/00/00 80A32463

UTTL: A comparison of computer architectures for the NASA demonstration advanced avionics system

AUTH: A/SEACORD, C. L.; B/BAILEY, D. G.; C/LARSON, J. C. PAA: C/(Honeywell, Inc., Avionics Div., Minneapolis, Minn.) CORP: Honeywell, Inc., Minneapolis, Minn. In: Challenge of the '80s; Proceedings of the Third Digital Avionics Systems Conference, Fort Worth, Tex., November 6-8, 1979. (A80-32417 12-06) New York, Institute of Electrical and Electronics Engineers, Inc., 1979, p. 51-57.

ABS: The paper compares computer architectures for the NASA demonstration advanced avionics system. Two computer architectures are described with an unusual approach to fault tolerance: a single spare processor can correct for faults in any of the distributed processors by taking on the role of a failed module. It was shown the system must be used from a functional point of view to properly apply redundancy and achieve fault tolerance and ultra reliability. Data are presented on complexity and mission failure probability which show that the revised version offers equivalent mission reliability at lower cost as measured by hardware and software complexity. 79/00/00 80A32427

UTTL: Wind power

AUTH: A/SIMMONS, D. M. Park Ridge, N.J., Noyes Data Corp. (Energy Technology Review, No. 6), 1975. 316 p.

ABS: The state-of-the art of wind conversion and storage system and wind machine design is reviewed. The properties of wind, based on the results of numerous wind studies, are discussed together with method of wind measurement and the selection of sites for wind power systems. Wind power research and development in the United States, Canada, the USSR, Germany, Denmark, France, Great Britain, Sweden, and several African and Asian countries is summarized. Commercially available wind power equipment and wind machine designs are described. 75/00/00 76A27784

UTTL: Validation of multiprocessor systems

AUTH: A/SIFWIOREK, D. P.; B/SEGALL, Z.; C/YORK, G.; D/KONG, T.; E/WILSON, A.; F/REILLY, M. CORP: Carnegie Mellon Univ., Pittsburgh, Pa. CSS: (Dept. of Computer Science and Electrical Engineering)

RPT#: NASA-CR-172687 NAS 1.26:172687 83/04/30 83N76330

UTTL: A fault tolerant multiprocessor architecture for aircraft, volume 2
AUTH: A/SMITH, T. B.; B/HOPKINS, A. L.; C/HALL, E. C.; D/HOWATT, J. R.; E/LALE, J. H. CORP: Draper (Charles Stark) Lab., Inc., Cambridge, Mass.
RPT#: NASA-CR-165915 NAS 1.26:165915 77/04/00 83N73266

UTTL: Survivable avionics computer system
AUTH: A/MONSON, P. R.; B/MONSON, C. A.; C/PEASE, M. C.; D/WISCHMEYER, C. E. CORP: SRI International Corp., Menlo Park, Calif. CSS: (Systems Techniques Lab.)
RPT#: AD-A111225 80/11/00 82N74287

UTTL: Digital avionics design for validation
AUTH: A/HITT, E. F. CORP: Battelle Columbus Labs., Ohio. In ASD Proc. Papers of the 2nd AFSC Avionics Std. Conf., Vol. 2 p 729-749 (SEE N84-31165 21-06) Proc. held in Dayton, Ohio, 30 Nov. - 2 Dec. 1982
ABS: The designer/developer of fault tolerant avionics must consider the requirements to validate these digital systems. These requirements should be primary factors influencing the design and hence the sustainability of these systems. This paper presents a synopsis of a methodology of design and validation of digital avionics and flight control systems based upon early consideration of validation requirements. Avionics developed using this methodology will provide real time fault detection and isolation and hence reduce aircraft down time due to avionics failures. Changes in mission requirements will be reflected in the need to modify or add software modules throughout the system's life cycle. This necessitates design and control of hardware and software interfaces in order to keep the time required to validate the change to a minimum and speed retrofit of the modification in the operational units.
RPT#: AD-P003571 82/11/00 84N31176

UTTL: Achieving the benefits of modular avionics design
AUTH: A/BEHNEN, S. W.; B/LIGHTFOOT, F. M.; C/METZ, P. R. CORP: Boeing Military Airplane Development, Seattle, Wash. In ASD Proc. Papers of the 2nd AFSC Avionics Std. Conf., Vol. 2 p 583-595 (SEE N84-31165 21-06) Proc. held in Dayton, Ohio, 30 Nov. - 2 Dec. 1982
ABS: New system development programs are adopting the principles of modular design to reduce the number of unique parts, increase capability, improve fault tolerance, lower costs, and encourage transition to new technologies. Programs such as Integrated Communication, Navigation, Identification Avionics are proving the value of this approach. Even greater benefits will be obtained from modular design, however, when the use of common modules spreads across, and into, dissimilar subsystems. The creation and adoption of new military standards, which will complement existing standards, are needed to encourage widespread use of compatible modular avionics. Examples are given, as well as suggestions as to the most efficient way of circumventing inherent industry reluctance to adopt national standards.
RPT#: AD-P003562 82/11/00 84N31167

UTTL: Standards and integrated avionic digital system architecture
AUTH: A/GRIFFIN, E. L. CORP: Martin Marietta Aerospace, Orlando, Fla. In ASD Proc. Papers of the 2nd AFSC Avionics Std. Conf., Vol. 2 p 563-581 (SEE N84-31165 21-06) Proc. held in Dayton, Ohio, 30 Nov. - 2 Dec. 1982
ABS: Integrated digital system design and development of the hardware, software, and interfaces that integrate the avionic flight control, fire control, and man-machine display and control must emphasize the man-rated weapon system's availability and survivability. The scope of tasks including detailed trade studies such as CMOS/SOS versus ECL semiconductor use, and parallel pipelining versus multi-microprocessor architecture usually requires an engineering team with backgrounds from requirements and integration, electronics hardware, packaging, and software. System attributes of fault tolerance, fail safe, and fail soft operation requires total team adherence to a set of design, documentation, implementation, and test standards of which few have complete familiarity. Since use of these standards has prevented costly errors and overruns in procurement, and decreased maintenance costs over the life cycle, this paper shows how to make each effective contributor on the team understand the standards controlling performance and product specifications, change and configuration control, test planning, and test procedure generation for the other areas of expertise.
RPT#: AD-P003561 82/11/00 84N31166

UTTL: Distributed avionic computers
AUTH: A/LEVITT, K. N.; B/MELLIAR-SMITH, P. M.; C/SCHWARTZ, R. L. CORP: SRI International Corp., Menlo Park, Calif.
ABS: The goal of the task described in this report is to establish the basis for an advanced fault tolerant onboard computer that will be the successor to the

current generation of fault-tolerant computers. Towards this goal we are working on the following technical problems: (1) The architecture of a network-based fault-tolerant system; (2) The diagnosis of transient faults from error reports; (3) The use of Ada as the language for the executives of the computer and the application programs; and (4) A new paradigm (an extension of the conventional voting paradigm) for comparing the values produced by replicated processors.

RPT#: AD-A142140 83/10/00 84N29494

UTTL: AFTI/F-16 digital flight control system experience

AUTH: A/MACKALL, D. A. CORP: National Aeronautics and Space Administration. Hugh L. Dryden Flight Research Center, Edwards, Calif.; National Aeronautics and Space Administration. Ames Research Center, Moffett Field, Calif. In NASA. Langley Research Center NASA Aircraft Controls Research, 1983 p 469-487 (SEE N84-20567 11-08)

ABS: The Advanced Flighter Technology Integration (AFTI) F-16 program is investigating the integration of emerging technologies into an advanced fighter aircraft. The three major technologies involved are the triplex digital flight control system; decoupled aircraft flight control; and integration of avionics, pilot displays, and flight control. In addition to investigating improvements in fighter performance, the AFTI/F-16 program provides a look at generic problems facing highly integrated, flight-crucial digital controls. An overview of the AFTI/F-16 systems is followed by a summary of flight test experience and recommendations. 84/03/00 84N20592

UTTL: Software testing of safety critical systems

AUTH: A/STOCKER, J. CORP: Messerschmitt-Boelkow-Blohm G.m.b.H., Munich (West Germany). CSS: (Aircraft Div.) In AGARD Advanced Concepts for Avionics/Weapon System Design, Develop. and Integration 9 p (SEE N84-15034 06-01)

ABS: Safety critical systems must be thoroughly tested. Powerful test facilities and test concepts are very important. The methods used for testing the software of the Tornado Autopilot and flight director system (AFDS) are outlined and existing test systems are reviewed followed by a presentation of a new test facility i.e., the AFDS Cross Software Test System (AFDS-CSTS). 83/10/00 84N15073

UTTL: System architecture: Key to future avionics capabilities

AUTH: A/ENGLAND, G. R. CORP: General Dynamics Corp., Fort Worth, Tex. CSS: (Avionic Systems Dept.) In AGARD Advanced Concepts for Avionics/Weapon System Design, Develop. and Integration 6 p (SEE N84-15034 06-01)

ABS: Modern aircraft still rely, as old World War II aircraft, upon the pilot or crew for integration of information from diverse discrete subsystems, sensors and weapons. During this long period of technology time, each generation of systems has generally become more complex and increased the quantity and rate of information to the crew. In many weapon system implementations these two factors have resulted in increased problems in the areas of system availability, affordability, supportability and operability. Although the F-16 has broken this trend it is still evident that new system architectures will be needed as mission requirements in the future create added system demands. Traditional avionic design, support and operability approaches will be unable to cope. The size reduction and performance improvements resulting from large scale and high speed integrated circuits will make it possible to restructure the way avionics systems are designed. For example, standard modules for multiuse applications are possible These modules become the building blocks for a new type of system architecture. Advanced data switching communication techniques provides the necessary data transfer rates to support sensor fusion, cockpit automation, and fault tolerant processing. Generic signal processors make shared functions realizable. 83/10/00 84N15035

UTTL: Fault tolerant architectures for integrated aircraft electronics systems

AUTH: A/LEVITT, K. N.; B/MELLIAR-SMITH, P. M.; C/SCHWARTZ, R. L. CORP: SRI International Corp., Menlo Park, Calif. NASA. Langley Research Center

ABS: Work into possible architectures for future flight control computer systems is described. Ada for Fault-Tolerant Systems, the NETS Network Error-Tolerant System architecture, and voting in asynchronous systems are covered.

RPT#: NASA-CR-172226 NAS 1.26:172226 83/08/00 84N10769

UTTL: Demonstration Advanced Avionics System (DAAS). Phase 1 report CORP: Honeywell, Inc., Minneapolis, Minn.; King Radio Corp., Olathe, Kans. CSS: (Avionics Div.)

ABS: An integrated avionics system which provides expanded functional capabilities that significantly enhance the

utility and safety of general aviation at a cost commensurate with the general aviation market is discussed. Displays and control were designed so that the pilot can use the system after minimum training Functional and hardware descriptions, operational evaluation and failure modes effects analysis are included.

RPT#: NASA-CR-170317 NAS 1.26:170317 N-151871 81/04/30 83N24504

UTTL: Software fault tolerance for real-time avionics systems

AUTH: A/ANDERSON, T.; B/KNIGHT, J. C. PAA: A/(Newcastle upon Tyne Univ.) CORP: National Aeronautics and Space Administration. Langley Research Center, Hampton, Va. In AGARD Software for Avionics 6 p (SEE N83-22112 12-01)

ABS: Avionics systems have very high reliability requirements and are therefore prime candidates for the inclusion of fault tolerance techniques. In order to provide tolerance to software faults, some form of state restoration is usually advocated as a means of recovery. State restoration can be very expensive for systems which utilize concurrent processes. The concurrency present in most avionics systems and the further difficulties introduced by timing constraints imply that providing tolerance for software faults may be inordinately expensive or complex. A straightforward pragmatic approach to software fault tolerance which is believed to be applicable to many real-time avionics systems is proposed. A classification system for software errors is presented together with approaches to recovery and continued service for each error type. 83/01/00 83N22132

UTTL: Validation of multiprocessor systems

AUTH: A/SIEWIOREK, D. P.; B/SEGALL, Z.; C/KONG, T. CORP: Carnegie-Mellon Univ., Pittsburgh, Pa. CSS: (Dept. of Computer Science.)

ABS: Experiments that can be used to validate fault free performance of multiprocessor systems in aerospace systems integrating flight controls and avionics are discussed. Engineering prototypes for two fault tolerant multiprocessors are tested.

RPT#: NASA-CR-169667 NAS 1.26:169667 82/12/23 83N14952

UTTL: CARE 3 phase 2 report - mathematical description

AUTH: A/STIFFLER, J. J.; B/BRYANT, L. A. CORP: Raytheon Co., Sudbury, Mass. NASA

ABS: CARE III (Computer-Aided Reliability Estimation, version three) a computer program designed to help estimate the reliability of complex, redundant systems is described. Although the program can model a wide variety of redundant structures, it was developed specifically for fault tolerant avionics systems. CARE III generalizes the class of system structures that can be modeled and greatly expands the coverage model to take into account such effects as intermittent and transient faults, latent faults, and error propagation.

RPT#: NASA-CR-3566 NAS 1.26:3566 82/11/00 83N14929

UTTL: The CARE 3 phase 3 report: Test and evaluation

AUTH: A/STIFFLER, J. J.; B/NEUMANN, J. S.; C/BRYANT, L. A. CORP: Raytheon Co., Sudbury, Mass.

ABS: CARE 3 (Computer-Aided Reliability Estimation, version three) is a computer program designed to help estimate the reliability of complex, redundant systems; although the program can model a wide variety of redundant structures, it was developed specifically for fault-tolerant avionics systems, systems distinguished by the need for extremely reliable performance since a system failure could well result in the loss of human life. CARE 3 further generalizes the class of system structures that can be modeled and greatly expands the coverage model to take into account such effects as intermittent and transient faults, latent faults, error propagation, etc. The initial test and evaluation of CARE 3 are reported.

RPT#: NASA-CR-3631 NAS 1.26:3631 82/11/00 83N12865

UTTL: The cost of software fault tolerance

AUTH: A/MIGNEAULT, G. E. CORP: National Aeronautics and Space Administration. Langley Research Center, Hampton, Va. Presented at the AGARD Avionics Panel Fall Meeting, The Hague, 6-9 Sep. 1982

ABS: The proposed use of software fault tolerance techniques as a means of reducing software costs in avionics and as a means of addressing the issue of system unreliability due to faults in software is examined. A model is developed to provide a view of the relationships among cost, redundancy, and reliability which suggests strategies for software development and maintenance which are not conventional.

RPT#: NASA-TM-84546 NAS 1.15:84546 82/09/00 83N10806

UTTL: Problems related to the integration of fault tolerant aircraft electronic systems
AUTH: A/SANNISTER, J. A.; B/ADLAKHA, V.; C/TRIVEDI, K.; D/ALSPAUGH, T. A., JR. CORP: Research Triangle Inst., Research Triangle Park, N.C. CSS: (Systems and Measurements Div.)
ABS: Problems related to the design of the hardware for an integrated aircraft electronic system are considered. Taxonomies of concurrent systems are reviewed and a new taxonomy is proposed. An informal methodology intended to identify feasible regions of the taxonomic design space is described. Specific tools are recommended for use in the methodology. Based on the methodology, a preliminary strawman integrated fault tolerant aircraft electronic system is proposed. Next, problems related to the programming and control of inegrated aircraft electronic systems are discussed. Issues of system resource management, including the scheduling and allocation of real time periodic tasks in a multiprocessor environment, are treated in detail. The role of software design in integrated fault tolerant aircraft electronic systems is discussed. Conclusions and recommendations for further work are included.
RPT#: NASA-CR-165926 NAS 1.26:165926 RTI/2094/02-02F 82/06/00 82N29022

UTTL: Production of Reliable Flight Crucial Software: Validation Methods Research for Fault Tolerant Avionics and Control Systems Sub-Working Group Meeting
AUTH: A/DUNHAM, J. R.; B/KNIGHT, J. C. PAA: B/(Virginia Univ.) CORP: Research Triangle Inst., Research Triangle Park, N.C. CSS: (Systems and Measurements Div.) Meeting held at Research Triangle Park, N.C., 2-4 Nov. 1981
ABS: The state of the art in the production of crucial software for flight control applications was addressed. The association between reliability metrics and software is considered. Thirteen software development projects are discussed. A short term need for research in the areas of tool development and software fault tolerance was indicated. For the long term, research in format verification or proof methods was recommended. Formal specification and software reliability modeling, were recommended as topics for both short and long term research.
RPT#: NASA-CP-2222 L-15291 NAS 1.55:2222 82/05/00 82N24845

UTTL: A bibliography on formal methods for system specification, design and validation
AUTH: A/MEYER, J. F.; B/FURCHTGOTT, D. G.; C/MOVAGHAR, A. CORP: Michigan Univ., Ann Arbor. CSS: (Systems Engineering Lab.)
ABS: Literature on the specification, design, verification, testing, and evaluation of avionics systems was surveyed, providing 655 citations. Journal papers, conference papers, and technical reports are included. Manual and computer-based methods were employed. Keywords used in the online search are listed.
RPT#: NASA-CR-168849 NAS 1.26:168849 REPT-014524-19-T UMICH-SEL-163 82/01/00 82N23990

UTTL: Analysis of computing system configurations for highly integrated guidance and control systems
AUTH: A/JONES, J. E.; B/KARMARKAR, J. S.; C/FLEMING, R. E. CORP: Systems Control, Inc., Palo Alto, Calif. In AGARD Guidance and Control Technol. for Highly Integrated Systems 9 p (SEE N82-23182 14-01)
ABS: The importance of early and sustained validation of architectures for highly integrated systems is discussed. Two early validation tools are presented. A description of the two tools, (1) generalized reliability and maintainability program (GRAMP), and (2) functional emulation, is presented, along with a discussion of their utility in the development of highly integrated guidance and control systems.
82/02/00 82N23189

UTTL: Integrated control of mechanical system for future combat aircraft
AUTH: A/WILCOCK, G. W.; B/LANCASTER, P. A.; C/MOXEY, C. CORP: Royal Aircraft Establishment, Farnborough (England).; British Aerospace Aircraft Group, Warton (England). In AGARD Tactical Airborne Distributed Computing and Networks 16 p (SEE N82-17086 08-01)
ABS: Various techniques for the application of digital control to aircraft utility systems were investigated. It is shown that the prefered approach utilizes a number of distributed processors and terminals that interface with the utility components. Analysis performed to data shows a weight saving of approximately 100 Kg (i.e. 50%), and a pilot workload reduction of the order of 4:1, may be achieved in a twin engine combat aircraft. 81/10/00 82N17117

UTTL: SIFT: An ultra-reliable avionic computing system

AUTH: A/MOSES, K. CORP: Bendix Corp., Teterboro, N.J. CSS: (Flight Systems Div.) In AGARD Tactical Airborne Distributed Computing and Networks 10 p (SEE N82-17086 08-01)

ABS: Software implemented fault tolerance (SIFT) is an ultra-reliable computing system which is based on a multiprocessor architecture that achieves fault tolerance by replicating computing tasks among processing units. Error detection and system configuration are performed by software to maintain the operational integrity of the computing system. The high speed inter-computer communication system required for operation realized by dedicated serial links arrayed in a star connection. Software algorithms are used for failure detection by means of voting, failure isolation to the faulty processor, and reconfiguration after fault detection. Frame synchronization between processors is employed to reduce data skew and minimize false alarms. The architecture of SIFT, its hardware implementation, and the test stand used for evaluation. Potential applications of this technique to current and anticipated ultra-reliable electrical flight control systems are given. 81/10/00 82N17115

UTTL: Dispersed sensor processing mesh project

AUTH: A/MEGNA, V. A. CORP: Draper (Charles Stark) Lab., Inc., Cambridge, Mass. In AGARD Tactical Airborne Distributed Computing and Networks 14 p (SEE N82-17086 08-01)

ABS: The F-8 dispersed sensor processing mesh project is an exploratory program involved in the development and test of the concept of a network communication structure. The elements of the structure are a bus controller and a number of nodes all of which are interconnected by multiple data flow paths. This structure is proposed as the communication medium between the subsystems of a distributed avionic system. In order to test and establish a data base for this proposed communication structure, the elements that comprise it must be designed and built. These elements are not just the hardware that is inherent to the structure, they also include the algorithms mechanized in the bus controller's software, the operating characteristic of the network, and the communication protocol used. The design and associated decisions made during the development of the network hardware and software are considered. 81/10/00 82N17113

UTTL: Toward the assessment of the susceptibility of a digital system to lightning upset

AUTH: A/TRONT, J. G.; B/GRAF, W.; C/MARTIN, C. F.; D/MASSON, G. M.; E/MYERS, J. M.; F/WHALEN, J. J. CORP: Virginia Polytechnic Inst. and State Univ., Blacksburg. CSS: (Dept. of Electrical Engineering)

ABS: Accomplishments and directions for further research aimed at developing methods for assessing a candidate design of an avionic computer with respect to susceptability to lightning upset are reported. Emphasis is on fault tolerant computers. Both lightning stress and shielding are covered in a review of the electromagnetic environment. Stress characterization, system characterization, upset detection, and positive and negative design features are considered. A first cut theory of comparing candidate designs is presented including tests of comparative susceptability as well as its analysis and simulation. An approach to lightning induced transient fault effects is included. 81/12/00 82N15295

UTTL: Measurement of fault latency in a digital avionic miniprocessor

AUTH: A/MCGOUGH, J. G.; B/SWERN, F. L. CORP: Bendix Corp., Teterboro, N.J. CSS: (Flight Systems Div.)

ABS: The results of fault injection experiments utilizing a gate-level emulation of the central processor unit of the Bendix BDX-930 digital computer are presented. The failure detection coverage of comparison-monitoring and a typical avionics CPU self-test program was determined. The specific tasks and experiments included: (1) inject randomly selected gate-level and pin-level faults and emulate six software programs using comparison-monitoring to detect the faults; (2) based upon the derived empirical data develop and validate a model of fault latency that will forecast a software program's detecting ability; (3) given a typical avionics self-test program, inject randomly selected faults at both the gate level and pin-level and determine the proportion of faults detected; (4) determine why faults were undetected; (5) recommend how the emulation can be extended to multiprocessor systems such as SIFT; and (6) determine the proportion of faults detected by a uniprocessor BIT (built-in-test) irrespective of self test

RPT# NASA CR-3462 81/10/00 82N10723

UTTL: Definition, analysis and development of an optical data distribution network for integrated avionics and control systems
AUTH: A/BURNS, R. R. CORP: Hughes Research Labs., Malibu, Calif.
ABS: The potential and functional requirements of fiber optic bus designs for next generation aircraft are assessed. State-of-the-art component evaluations and projections were used in the system study. Complex networks were decomposed into dedicated structures, star buses, and serial buses for detailed analysis. Comparisons of dedicated links, star buses, and serial buses with and without full duplex operation and with considerations for terminal to terminal communication requirements were obtained. This baseline was then used to consider potential extensions of busing methods to include wavelength multiplexing and optical switches. Example buses were illustrated for various areas of the aircraft as potential starting points for more detail analysis as the platform becomes definitized.
RPT#: NASA-CR-159370 81/08/00 81N30962

UTTL: Fault tolerance design and redundancy management techniques CORP: Advisory Group for Aerospace Research and Development, Neuilly-Sur-Seine (France). Lecture Series held in Athens, 13-14 Oct. 1980; Rome, 16-17 Oct. 1980; and London, 20-21 Oct. 1980
RPT#: AGARD-LS-109 ISBN-92-835-0274-4 AD-A090849 80/09/00 81N11266

UTTL: Design and development of software for Sea Harrier HUDWAC
AUTH: A/JONES, E. P.; B/HOWISON, S. CORP: Smiths Industries Ltd., Bishops Cleeve (England). CSS: (Electronic Displays Branch.) In AGARD Guidance and Control Software 12 p (SEE N80-32125 22-61)
ABS: The development of the airborne software for the Sea Harrier Head-Up Display and Weapon Aiming Computer System is described. The system requirements are outlined and include the symbology generation for both head-up and head-down display systems; weapon aiming computations; flight path guidance for both target interception and aircraft to ship recovery; pointing and control commands for radar and guided weapons; first line avionic system test and fault identification; and extensive operational self test and failure tolerance. The hardware configuration, software design, and software writing utilizing the CORAL programming language are discussed. 80/05/00 80N32138

UTTL: Remote link unit functional design: An advanced remote terminal for MIL-STD-1553B
AUTH: A/TAVORA, C. J.; B/GLOVER, J. R., JR.; C/BATTEN, G W. CORP: Houston Univ., Tex. CSS: (Dept. of Electrical Engineering.)
ABS: This report presents a functional design of the Remote Link Unit (RLU). The RLU is an intelligent remote terminal which is compatible with hierarchical distributed processing. It has interface modules capable of processing subsystem data on-the-fly and supporting a universal signal interface which can be configured under software control to input or output analog and digital signals of varied types. Electronic nameplates containing identification, interface requirements and signal processing programs are attached to avionics subsystems to support the RLU operational features. The RLU stand-alone processing capabilities provide fault tolerance and facilitate maintenance.
RPT#: AD-A080126 AFAL-TR-79-1176 79/10/00 80N22558

UTTL: Distributed processor/memory architectures design program
AUTH: A/CONSOLVER, G.; B/ACKLEY, D.; C/RICKARD, M.; D/MCAFEE, R.; E/SHIPCHANDLER, T. CORP: Texas Instruments, Inc., Dallas.
ABS: The purpose of Distributed Processor/Memory Architecture (DP/M) design program was to extend the DP/M avionic system processing concept to a detailed system hardware and software design. The functional design for the DP/M Processing Element (PE) is summarized, including the processor, memory, input/output interface, and a dual-level time-division-multiplex bus interface unit. A set of simulation and analysis programs was developed for modeling both the high-level network interaction among interconnected Processing Elements and the detailed internal operation of the PE. Other major areas examined were the executive control software, the process construction methodology required to develop and allocate real-time software for DP/M, and methods that could be used with DP/M to promote avionic system fault tolerance.
RPT#: AD-A016482 AFAL-TR-75-80 75/02/00 76N20845

REPORT DOCUMENTATION PAGE

1. Recipient's Reference	2. Originator's Reference	3. Further Reference	4. Security Classification of Document
	AGARD-LS-143	ISBN 92-835-1510-2	UNCLASSIFIED

5. Originator Advisory Group for Aerospace Research and Development
North Atlantic Treaty Organization
7 rue Ancelle, 92200 Neuilly sur Seine, France

6. Title FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION

7. Presented on 1—2 October 1985 in Edwards, USA, 17—18 October 1985 Copenhagen, Denmark, and 21—22 October 1985 in Athens, Greece.

8. Author(s)/Editor(s)	9. Date
Various	September 1985
10. Author's/Editor's Address	**11. Pages**
Various	150

12. Distribution Statement This document is distributed in accordance with AGARD policies and regulations, which are outlined on the Outside Back Covers of all AGARD publications.

13. Keywords/Descriptors

Computer systems programs
Computer systems hardware
Flight control
Errors
Tolerances (mechanics)

14. Abstract

This Lecture Series is intended to provide basic concepts and theories in the design of fault-tolerant architectures for flight critical systems. It is intended to cover experience with flight tested fly-by-wire systems as well as issues in redundancy management of synchronous and asynchronous approaches. It will specifically address the individual aspects of software fault tolerance, actuation fault tolerance, reliable data communication, and multi-computer operation using the ADA language. Partial contents: ... (iv)

This Lecture Series, sponsored by the Guidance and Control Panel, has been implemented by the Consultant and Exchange Programme of AGARD.

AGARD Lecture Series No.143 Advisory Group for Aerospace Research and Development, NATO FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION Published September 1985 150 pages This Lecture Series is intended to provide basic concepts and theories in the design of fault-tolerant architectures for flight critical systems. It is intended to cover experience with flight tested fly-by-wire systems as well as issues in redundancy management of synchronous and asynchronous approaches. It will specifically address the individual aspects of software fault tolerance, actuation P.T.O.	AGARD-LS-143 Computer systems programs Computer systems hardware Flight control Errors Tolerances (mechanics)	AGARD Lecture Series No.143 Advisory Group for Aerospace Research and Development, NATO FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION Published September 1985 150 pages This Lecture Series is intended to provide basic concepts and theories in the design of fault-tolerant architectures for flight critical systems. It is intended to cover experience with flight tested fly-by-wire systems as well as issues in redundancy management of synchronous and asynchronous approaches. It will specifically address the individual aspects of software fault tolerance, actuation P.T.O.	AGARD-LS-143 Computer systems programs Computer systems hardware Flight control Errors Tolerances (mechanics)
AGARD Lecture Series No.143 Advisory Group for Aerospace Research and Development, NATO FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION Published September 1985 150 pages This Lecture Series is intended to provide basic concepts and theories in the design of fault-tolerant architectures for flight critical systems. It is intended to cover experience with flight tested fly-by-wire systems as well as issues in redundancy management of synchronous and asynchronous approaches. It will specifically address the individual aspects of software fault tolerance, actuation P.T.O.	AGARD-LS-143 Computer systems programs Computer systems hardware Flight control Errors Tolerances (mechanics)	AGARD Lecture Series No.143 Advisory Group for Aerospace Research and Development, NATO FAULT TOLERANT HARDWARE/SOFTWARE ARCHITECTURE FOR FLIGHT CRITICAL FUNCTION Published September 1985 150 pages This Lecture Series is intended to provide basic concepts and theories in the design of fault-tolerant architectures for flight critical systems. It is intended to cover experience with flight tested fly-by-wire systems as well as issues in redundancy management of synchronous and asynchronous approaches. It will specifically address the individual aspects of software fault tolerance, actuation P.T.O.	AGARD-LS-143 Computer systems programs Computer systems hardware Flight control Errors Tolerances (mechanics)

fault tolerance, reliable data communication, and multi-computer operation using the ADA language. The material in this publication was assembled to support a Lecture Series under the sponsorship of the Guidance and Control Panel and the Consultant and Exchange Programme of AGARD presented on 1—2 October 1985 in Edwards, USA, 17—18 October 1985 in Copenhagen, Denmark, and 21—22 October 1985 in Athens, Greece. ISBN92-835-1510-2	fault tolerance, reliable data communication, and multi-computer operation using the ADA language. The material in this publication was assembled to support a Lecture Series under the sponsorship of the Guidance and Control Panel and the Consultant and Exchange Programme of AGARD presented on 1—2 October 1985 in Edwards, USA, 17—18 October 1985 in Copenhagen, Denmark, and 21—22 October 1985 in Athens, Greece. ISBN92-835-1510-2
fault tolerance, reliable data communication, and multi-computer operation using the ADA language. The material in this publication was assembled to support a Lecture Series under the sponsorship of the Guidance and Control Panel and the Consultant and Exchange Programme of AGARD presented on 1—2 October 1985 in Edwards, USA, 17—18 October 1985 in Copenhagen, Denmark, and 21—22 October 1985 in Athens, Greece. ISBN92-835-1510-2	fault tolerance, reliable data communication, and multi-computer operation using the ADA language. The material in this publication was assembled to support a Lecture Series under the sponsorship of the Guidance and Control Panel and the Consultant and Exchange Programme of AGARD presented on 1—2 October 1985 in Edwards, USA, 17—18 October 1985 in Copenhagen, Denmark, and 21—22 October 1985 in Athens, Greece. ISBN92-835-1510-2

AGARD

NATO OTAN

7 RUE ANCELLE · 92200 NEUILLY-SUR-SEINE

FRANCE

Telephone 745.08.10 · Telex 610176

DISTRIBUTION OF UNCLASSIFIED AGARD PUBLICATIONS

AGARD does NOT hold stocks of AGARD publications at the above address for general distribution. Initial distribution of AGARD publications is made to AGARD Member Nations through the following National Distribution Centres.Further copies are sometimes available from these Centres, but if not may be purchased in Mircrofiche or Photocopy form from the Purchase Agencies listed below

NATIONAL DISTRIBUTION CENTRES

BELGIUM
Coordonnateur AGARD – VSL
Etat-Major de la Force Aerienne
Quartier Reine Elisabeth
Rue d'Evere, 1140 Bruxelles

CANADA
Defence Scientific Information Services
Dept of National Defence
Ottawa, Ontario K1A 0K2

DENMARK
Danish Defence Research Board
Ved Idraetsparken 4
2100 Copenhagen Ø

FRANCE
O.N.E.R.A. (Direction)
29 Avenue de la Division Leclerc
92320 Châtillon

GERMANY
Fachinformationszentrum Energie,
Physik, Mathematik GmbH
Kernforschungszentrum
D-7514 Eggenstein-Leopoldshafen

GREECE
Hellenic Air Force General Staff
Research and Development Directorate
Holargos, Athens

ICELAND
Director of Aviation
c/o Flugrad
Reyjavik

ITALY
Aeronautica Militare
Ufficio del Delegato Nazionale all'AGARD
3 Piazzale Adenauer
00144 Roma/EUR

LUXEMBOURG
See Belgium

NETHERLANDS
Netherlands Delegation to AGARD
National Aerospace Laboratory, NLR
P.O. Box 126
2600 AC Delft

NORWAY
Norwegian Defence Research Establishment
Attn: Biblioteket
P.O. Box 25
N-2007 Kjeller

PORTUGAL
Portuguese National Coordinator to AGARD
Gabinete de Estudos e Programas
CLAFA
Base de Alfragide
Alfragide
2700 Amadora

TURKEY
Department of Research and Development (ARGE)
Ministry of National Defence, Ankara

UNITED KINGDOM
Defence Research Information Centre
Station Square House
St Mary Cray
Orpington, Kent BR5 3RE

UNITED STATES
National Aeronautics and Space Administration (NASA)
Langley Research Center
M/S 180
Hampton, Virginia 23665

THE UNITED STATES NATIONAL DISTRIBUTION CENTRE (NASA) DOES NOT HOLD STOCKS OF AGARD PUBLICATIONS, AND APPLICATIONS FOR COPIES SHOULD BE MADE DIRECT TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS) AT THE ADDRESS BELOW.

PURCHASE AGENCIES

Microfiche or Photocopy

National Technical
Information Service (NTIS)
5285 Port Royal Road
Springfield
Virginia 22161, USA

Microfiche

ESA/Information Retrieval Service
European Space Agency
10, rue Mario Nikis
75015 Paris, France

Microfiche or Photocopy

British Library Lending
Division
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
England

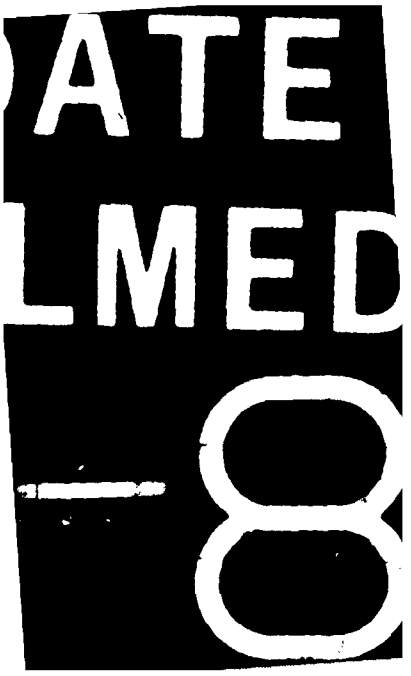
Requests for microfiche or photocopies of AGARD documents should include the AGARD serial number, title, author or editor, and publication date. Requests to NTIS should include the NASA accession report number. Full bibliographical references and abstracts of AGARD publications are given in the following journals:

Scientific and Technical Aerospace Reports (STAR)
published by NASA Scientific and Technical
Information Branch
NASA Headquarters (NIT-40)
Washington D.C. 20546, USA

Government Reports Announcements (GRA)
published by the National Technical
Information Services, Springfield
Virginia 22161, USA

Printed by Specialised Printing Services Limited
40 Chigwell Lane, Loughton, Essex IG10 3TZ

ISBN 92-835-1510-2


ATE
LMED
8